McGRAW-HILL

32 BITS

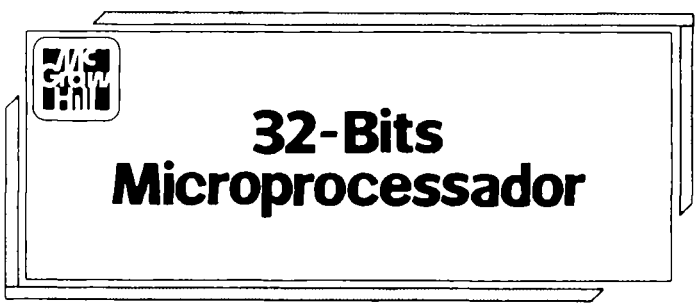
MICROPROCESSADOR

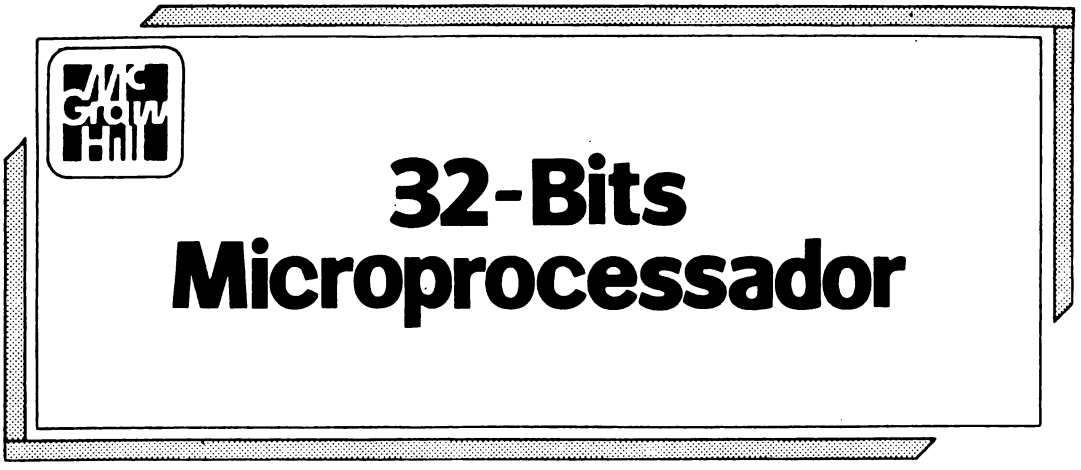
INTEL 80386
MOTOROLA 68020
AT&T WE32100
ZILOG 280000

McGraw-Hill

H.J.MITCHELL

LITEC
LIVRARIA EDITORA TÉCNICA LTDA
Rua dos Timbiras, 257 - CEP 01208 - São Paulo
Tel 222-0477 - Fax 220-2058
REF. 00900 PREÇO





32-Bits Microprocessador

H. J. Mitchell
Coordenador

Tradução

Maria Isabel Perez Otero
Analista de Software

Revisão Técnica

João Hajime Takeda
Pesquisador do Laboratório de Sistemas Integrados (LSI) EPUSP

McGraw-Hill
São Paulo
Rua Tabapuã, 1.105, Itaim-Bibi
CEP 04533
(011) 881-8604 e (011) 881-8528

Rio de Janeiro • *Lisboa* • *Porto* • *Bogotá* • *Buenos Aires* • *Guatemala* • *Madrid* • *México* • *New York* • *Panamá* • *San Juan* • *Santiago*

Auckland • Hamburg • Kuala Lumpur • London • Milan • Montreal • New Delhi • Paris • Singapore • Sydney • Tokyo • Toronto

Do original
32-BIT Microprocessors

Copyright © 1986 by H. J. Mitchell
Copyright © 1988 da Editora McGraw-Hill, Ltda.

Todos os direitos para a língua portuguesa reservados pela Editora McGraw-Hill, Ltda.

Nenhuma parte desta publicação poderá ser reproduzida, guardada pelo sistema "retrieval" ou transmitida de qualquer modo ou por qualquer outro meio, seja este eletrônico, mecânico, de fotocópia, de gravação, ou outros, sem prévia autorização, por escrito, da Editora.

Editor: Milton Mira de Assumpção Filho
Coordenadora de Revisão: Daisy Pereira Daniel
Supervisor de Produção: José Rodrigues

Dados de Catalogação na Publicação (CIP) Internacional
(Câmara Brasileira do Livro, SP, Brasil)

T754	32-Bits microprocessador / H.J. Mitchell, coordenador ; tradução Maria Isabel Perez Otero ; revisão técnica João Hajime Takeda. -- São Paulo : McGraw-Hill, 1988. 1. Microprocessadores I. Mitchell, H.J.
88-0110	CDD-001.6404

Índices para catálogo sistemático:

1. Microprocessadores : Processamento de dados 001.6404
2. Microprocessadores 32 bits : Processamento de dados 001.6404

SUMÁRIO

PREFÁCIO

O aumento do número de microprocessadores de 32 bits levou-nos a elaborar este livro, cujo objetivo é descrever em detalhes um conjunto representativo desses novos dispositivos.

Em qualquer livro semelhante a este, o autor é movido pelo desejo de organização para prover, em alguns casos, informações preliminares e o esforço em preparar seus capítulos. Como nunca será possível concluir um livro se se tentar abranger todos os dispositivos, foi necessário impor algum tipo de limitação com relação ao número de contribuições e o tempo dedicado aos colaboradores. Embora haja omissões, o livro oferece uma visão balanceada do que está atualmente presente, o que está vindo e o que virá no futuro.

O livro procurou prover informação detalhada sobre a arquitetura e a operação dos microprocessadores de 32 bits analisados. A informação apresenta-se em nível aceitável para um projetista de sistema, para um engenheiro e para um estudante de pesquisa. Hoje um microprocessador não pode ser considerado isoladamente, mas sim juntamente com seus componentes de hardware e software e as ferramentas necessárias para desenvolver um sistema.

O primeiro capítulo é uma discussão de RISC, que agora está se deslocando dos meios acadêmicos para os industriais. Com ambos, microprocessadores e sistemas, surgindo, usando essa filosofia, considerou-se apropriado inserir um artigo de abrangência geral sobre essa importante área.

Capítulos subseqüentes tratam de dispositivos específicos. Com exceção do capítulo sobre o Transputer da Inmos, todas as contribuições são dadas pelos fabricantes dos dispositivos.

CAPÍTULO 1

INTRODUÇÃO

H. J. Mitchell
CAP SCIENTIFIC Ltd

Para os que têm trabalhado com microcomputadores desde seu aparecimento, o progresso para 32 bits tem sido uma evolução consistente. Esses novos dispositivos têm sido cuidadosamente pesquisados e planejados, ao contrário do início dos anos 70, quando os dispositivos apareciam com tal rapidez que os usuários arriscavam-se a perder grandes investimentos devido a uma má escolha.

Nos anos 70 havia uma competição feroz no mercado de instrumentação levando ao emprego de microprocessadores em larga escala. Hoje, o microprocessador de 32 bits está ajudando a criar e expandir novos mercados em áreas desde *workstations* (estações de trabalho) até áreas esotéricas, como processamento de sinal em tempo real, onde matrizes de processadores se tornarão triviais. Mesmo os fabricantes de computadores estão desenvolvendo e vendendo seus próprios microprocessadores de 32 bits para produtos comuns, isso se o Micro VAX II puder ser chamado de um produto comum!

A maioria dos fabricantes de microprocessadores está mudando sua arquitetura de 16 para 32 bits e também aumentando suas funcionalidades. A principal razão desse enfoque parece ser a alegação de que estão protegendo o investimento do usuário em software. Para algumas aplicações isto é obviamente importante. Entretanto, para muitas aplicações novas, esse argumento é totalmente irrelevante; tem-se o IBM PC usando o Intel 8088 e 80286. O próximo passo lógico seria uma mudança para o 80386; entretanto reportagens recentes sugerem o contrário*. Da mesma maneira a Zilog parece estar planejando o uso do AT&T 31000 em vez do Z80000 para sua próxima geração de sistemas UNIX. Pode-se argumentar que alguns fabricantes de semicondutores estão tentando proteger mais seus próprios investimentos em desenvolvimento de sistemas de software

* *N.R.:* Apesar disso, há informes indicando tendência contrária da IBM, inclusive mostrando projetos e protótipos usando o 80386.

que os usuários. A principal área na qual a portabilidade de software é importante é no mercado de minicomputadores. Há alguns anos a DEC percebeu isso e produziu o PDP 11 e agora as máquinas VAX; outros, notavelmente a Hewlett-Packard, estão ainda tentando nivelar-se. A Hewlett-Packard está tentando com sua linha Spectrum unificar sua linha fragmentada de produtos com uma arquitetura comum cobrindo desde estações de trabalho até *mainframes* (computadores de grande porte).

O conceito de linha de produtos unificados é de senso comum; o que faz o trabalho da Hewlett-Packard interessante é a intenção de adotar um computador com reduzido conjunto de instruções (RISC – Reduced Instruction Set Computer) como arquitetura básica para produtos futuros. A Hewlett-Packard será a primeira grande fabricante de computadores ou semicondutores a adotar esse enfoque para todos os novos produtos. Entretanto, a IBM e outras estão tendo mais cautelas ao iniciar apenas a introdução de estações de trabalho usando arquitetura RISC. No lado oposto da escala, o Transputer da Inmos, com sua arquitetura RISC, tem o potencial de mudar a face da computação como a conhecemos hoje. Com dispositivos RISC provindos da Fairchild, Acorn e outras, sem dúvida, em breve, saber-se-á quais argumentações são válidas nos debates sobre RISC/CISC.

Provavelmente o nome mais comum neste livro será UNIX. Nos últimos anos UNIX tem aparecido como sistema operacional na maioria dos sistemas de computadores de pequeno a grande porte. Goste-se ou não, UNIX será usado ainda por vários anos. Reconhecendo isto, a maioria dos fabricantes de semicondutores tem projetado algumas características em seus dispositivos para suportar UNIX. Como se poderia esperar, o dispositivo da AT&T é projetado para suporte eficiente de linguagem C. Esse dispositivo tem também incluído algumas primitivas para melhorar a performance do UNIX. Dos poucos microprocessadores de 32 bits existentes no mercado, o sistema operacional UNIX tem aparecido em sistemas incorporando dispositivos Motorola, National Semicondutor e AT&T.

Enquanto UNIX é agora um dos sistemas operacionais padrão para aplicações de uso geral, não existe padronização para aplicações específicas. Embora o executivo RMX da Intel seja provavelmente um dos mais amplamente utilizados, deveríamos notar uma mudança vagarosa para o uso de Ada. No entanto, a menos que a qualidade e a performance desses primeiros computadores melhorem, muitos usuários relutarão em mudar de linguagem e sistemas operacionais já conhecidos. É também provável que as complexidades dessa linguagem e suas ferramentas de suporte possam deter muitas organizações em apostarem em desenvolvimento usando Ada.

Para sistemas mais complexos, o uso de multiprocessadores está se tornando comum. Percebendo isto, muitos fabricantes projetaram algum suporte em seus produtos para aplicações em multiprocessadores. Em particular a Inmos e a Zilog projetaram em seus produtos extensas características para tais aplicações. A mais nova dessas caracterís-

ticas é a encontrada no Transputer, da Inmos, com seus (atualmente) quatro enlaces (*links*) seriais de alta velocidade. Se se está realmente utilizando uma configuração de multiprocessadores ou co-processadores, as ferramentas básicas têm de ser projetadas e incorporadas no dispositivo no início, para garantir que essas facilidades estejam disponíveis ao usuário. Atualmente o tipo mais comum de co-processador é aquele para aplicações aritméticas ou de ponto flutuante. Apesar de esses dispositivos serem ainda mais lentos que os equivalentes em minicomputadores, melhoras podem ser esperadas.

Com usuários exigindo melhor desempenho de cada versão de produto e capacidades de memória cada vez maiores, o planejamento e controle de sistemas de memórias é hoje uma área complexa. Embora o suporte a gerenciamento de memória seja um requisito essencial a todos os microprocessadores de 32 bits, o projetista de sistema tem agora de considerar a possibilidade de utilização de memória cache externa. Com pouca memória cache aparecendo na maioria dos novos dispositivos, é provável que para alta performance ou sistemas com ampla utilização de memória, o uso dessa memória externa seja essencial. Para permitir aos sistemas operarem próximos de sua velocidade máxima, o projetista tem unicamente a escolha entre o elevado custo de memória estática de alta velocidade ou a implementação de memória cache para preceder memórias mais lentas, baseadas em memória dinâmica.

Pode-se assumir que em sua maioria os microprocessadores de 32 bits serão utilizados em projetos grandes e tecnicamente complexos. Esses projetos necessitarão de planejamento cuidadoso, recursos e suporte para atingirem seus objetivos. Qualquer projeto de médio a grande porte necessitará de ferramentas de suporte integradas e extensivas. Essas facilidades terão de incluir ferramental de gerenciamento, automação de escritório, utilitários para projetos de software e sistemas e também para suporte geral, como um sistema de gerenciamento de código. Se por um lado essas ferramentas estão se tornando comuns, a área onde há maior necessidade de trabalho é a de projeto de software e processo de controle. Com a confiabilidade do software aparecendo agora como cláusula contratual em projetos militares, todo o processo de controle do projeto tem de ser muito cuidadoso.

Com pelo menos quatro dispositivos atualmente disponíveis e muitos outros se apresentando no futuro próximo, pode-se dizer que a era do microprocessador de 32 bits chegou. Para onde a indústria se direcionará a seguir está nas mãos dos usuários, seus mercados e fabricantes de semicondutores. Se esse direcionamento será para o incremento da funcionalidade, aumento dos tamanhos de palavras, RISC, ou paralelismo, apenas o tempo dirá.

CAPÍTULO **2**

UMA PERSPECTIVA RISC

H. M. Brinkely Sprunt, E. Douglas Jensen,
Charles Y. Hitchcock III e Robert P. Colwell
Departamento de Ciência da Computação, Departamento de Engenharia Elétrica e de Computação, Carnegie-Mellon, University, Pittsburgh, Pa., USA

2.1 INTRODUÇÃO

Um efetivo projetista de computador deve tomar muitas decisões quando cria uma nova arquitetura de computador. Essas decisões são baseadas tanto na experiência do projetista como nas requisições do sistema. Recentemente o estilo de projeto de computador com conjunto de instruções reduzido (RISC) tem recebido muita atenção e muitos de seus proponentes crêem que a filosofia RISC oferece a melhor metodologia para projetar computadores. Declarações foram feitas, onde os projetos RISC produzem máquinas que são mais baratas e mais fáceis de projetar que computadores com conjunto de instruções complexo (CISC – Complex Instruction Set Computer). Entretanto, embora a filosofia RISC tenha méritos, as implicações desse estilo de projeto nao são bem compreendidas. A filosofia RISC propõe um sério desafio para várias suposições implícitas que têm guiado projetos de computadores há anos. Todavia, artigos de pesquisa sobre RISC freqüentemente falham na exploração apropriada de vários procedimentos e podem estar incorretos. Por exemplo, são feitas comparações entre máquinas RISC e CISC difíceis de interpretar, porque os dados sobre desempenho são monolíticos por natureza e não associam devidamente benefícios de desempenho apropriados aos mecanismos envolvidos. Têm sido feitas também comparações entre máquinas de implementações e objetivos de projeto bem diferentes (exemplo: RISC I e VAX), sem esclarecer essas diferenças, tornando os resultados ilusórios. O objetivo deste capítulo é oferecer uma perspectiva mais proveitosa do estudo RISC/SISC, uma pesquisa que é baseada nos resultados da experiência recente da Carnegie-Mellon University (CMU).

Embora nem todas as distorções e mal-entendidos possam ser corrigidos aqui, alguns deles estão no coração do projeto de computador e serão o enfoque deste capítulo. Por exemplo, grande parte da literatura RISC é voltada para discussões sobre o tamanho

e complexidade do conjunto de instruções do computador. Isto é extremamente enganoso. Projeto de conjunto de instruções é muito importante, mas não deveria ser dirigido somente pela aderência às convicções de estilo de projeto, RISC ou CISC. O foco das discussões poderia recair em questões mais gerais da associação da funcionalidade do sistema como o nível de implementação dentro da arquitetura. Esse ponto de vista inclui o conjunto de instruções (em geral, os CISC tendem a instalar funcionalidade em níveis mais baixos do sistema que o RISC), mas também considera outras características de projeto como conjunto de registradores, co-processadores e caches.

Ainda que as implicações do estudo sobre RISC se estendam ao conjunto de instruções, dentro desse domínio há limitações que não têm sido apresentadas. O exame de alguns artigos RISC típicos irá proporcionar algumas pistas, se houver, de onde a técnica RISC poderia falhar. Propostas de máquinas simples de alto desempenho são atrativas, mas, como se diz, "Todo problema complexo tem uma solução simples... e ela é errada". As idéias RISC não são "erradas", mas uma visão simplista delas seria.

A filosofia RISC tem várias implicações que não são óbvias. As pesquisas sobre RISC têm ajudado a focalizar a atenção em vários temas importantes de arquitetura de computadores cujas soluções têm sido, muito freqüentemente, decididas por *default*. Os proponentes de RISC, ainda assim, falham ao discutir as aplicações, arquitetura e contexto de implementação, nos quais suas afirmações parecem ser justificadas. Mesmo assim, uma solução cuidadosa do estilo de projeto RISC, suas propostas e os desafios que ele propõe às máquinas CISC podem produzir um conhecimento mais profundo dos compromissos hardware/software, desempenho de computadores, a influência do VLSI para projetos de processadores e vários outros tópicos.

Neste capítulo, vamos rapidamente rever o desenvolvimento em projetos de computadores bem como projetos de estudo sobre RISC, os quais levaram ao estilo de projeto RISC. Depois serão revistos alguns dos desafios significantes do RISC para projetos de computadores e discutidas várias máquinas comerciais RISC. Depois de apresentadas as informações históricas e alguns exemplos de máquinas RISC, serão discutidos, o que se pensa ser, os pontos de maior confusão e mal entendimentos associados com RISC. Essa discussão leva ao sumário do projeto de estudo conduzido pela CMU, o qual dá uma perspectiva melhor da controvérsia RISC/CISC.

2.2 ORIGEM DO RISC

Para compreender a motivação de um computador com conjunto de instruções reduzido é importante primeiro entender-se as tendências históricas do projeto de con-

junto de instruções. Desde os primeiros computadores eletrônicos digitais, os conjuntos de instruções tendem a ser maiores e mais complexos. O MARK-1, 1948, não tinha mais que sete instruções de complexidade trivial como soma e desvios simples; entretanto, uma máquina contemporânea, como o VAX, tem mais de trezentas instruções. Além disso, suas instruções podem ser particularmente complicadas, como inserção atômica de um elemento em uma lista duplamente ligada, ou a avaliação de um polinômio de grau arbitrário em ponto flutuante. Qualquer implementação de alto desempenho do VAX, como resultado, deve contar com técnicas complexas de implementação como *pipelining* (canalização de recursos), *prefetching* (pré-aquisição de instrução) e caches[31].

Esse tipo de progressão, de pequenos e simples para grandes e complexos conjuntos de instruções, é mais notável em processadores em uma única pastilha, pois seu desenvolvimento ocorreu apenas na década passada. O contraste entre o 6800 e o 68020, ambos da Motorola, por exemplo, mostra a adição de 11 modos de endereçamento, mais do dobro, como também várias instruções, e a adição de novas funcionalidades tais como suporte para cache de instruções e co-processadores. Novamente, não só o número de modos de endereçamento e instruções tem crescido, mas também suas complexidades.

Essa tendência geral para máquinas CISC foi alimentada por várias causas, incluindo as seguintes:

- freqüentemente é requisitado que novos modelos dentro de uma família de computadores sejam compatíveis com os existentes, resultando em "superconjuntos" e proliferação de características. Isto permite que características novas sejam incorporadas em máquinas novas sem alienar grandes bases de software;

- vários projetistas de computadores desejam reduzir a "lacuna semântica" entre programas e conjunto de instruções. Adicionando instruções semanticamente mais parecidas com as que o programador precisa, esses projetistas esperam reduzir o custo do software com o provento de uma máquina mais fácil de programar[8]. Note-se que tais instruções tendem a ser mais complexas devido ao nível semântico mais alto. (Entretanto, é freqüente o caso em que instruções com alto conteúdo semântico não atendem a necessidades requeridas por uma linguagem específica.)[33];

- no esforço para máquinas mais rápidas, os projetistas têm migrado de funções de software para microcódigo, ou de microcódigo para hardware. Mas freqüentemente isto é feito com insensibilidade aos efeitos perniciosos que poderão advir do incremento de uma característica à arquitetura. Por exemplo, implementando-se uma instrução que requer um nível adicional de decodificação lógica, todo o conjunto de instruções da máquina pode se tornar vagaroso (isto é chamado fenômeno 'n + 1')[10];

- ferramentas e metodologias têm ajudado os projetistas no tratamento de complexidades inerentes a grandes arquiteturas. As ferramentas CAD atuais e os programas de suporte para microcódigo são exemplos.

Microcódigo é um exemplo particular e interessante de uma técnica que tem encorajado projetos complexos, e isto ocorre de duas formas. Primeira, ele fornece um meio estruturado para criar e alterar algoritmos que controlam a execução de numerosas e complexas instruções. Segunda, a proliferação de características CISC é incentivada pela natureza quântica da memória de microcódigo. É relativamente fácil adicionar outro modo de endereçamento ou instrução obscura a uma máquina que ainda não esgotou seu espaço para microcódigo.

O rastreamento de instruções em máquinas CISC mostra consistentemente que a maioria das instruções disponíveis é usada com pouca freqüência em um dado ambiente de computação. Essa situação implica várias coisas, que serão vistas ainda neste capítulo. Uma observação semelhante das características do System/360 levou John Coke, da IBM, na década de 70, a considerar um desvio do estilo de computador tradicional. O resultado foi um projeto de pesquisa designado criativamente pelo número do edifício do grupo de pesquisa, 801, o qual foi baseado em alguns poucos princípios coerentes e sinergéticos de projeto. Pouco foi publicado sobre o projeto, porém o que foi narra um esforço de pesquisa coerente e de princípios.

Três idéias básicas nortearam o projeto de sistema 801:

- prover uma máquina que possa executar suas instruções em um ciclo de máquina e escolhê-las para serem um bom alvo para um compilador. Isto implica que instruções simples executadas freqüentemente não tenham seu desempenho penalizado por um hardware adicional necessário para gerenciar a execução de instruções mais complexas (que podem envolver decodificação mais extensiva e operações de vários ciclos);
- projetar a hierarquia de memória de forma que o engenho de computar não tenha de esperar por acessos à memória. O desempenho potencial de um computador simples e rápido poderia ser perdido se sua execução fosse interrompida freqüentemente pela espera por instruções ou dados;
- basear todo o projeto do sistema no uso profundo do compilador 801. Os programadores estão agora contando muito com compiladores para melhor desenvolvimento de programas e manejo de raciocínios, utilizando a linguagem assembly somente quando é essencial um desempenho ótimo ou para operações que não podem ser especificadas pela linguagem fonte.

Essas idéias resultaram em uma arquitetura de conjunto de instruções baseadas

em três princípios de projeto. De acordo com Radin[27], o conjunto de instruções era aquele conjunto de operações em tempo de processamento que:

- não podiam ser movidas para tempo de compilação;
- não podiam ser executadas mais eficientemente por código objeto produzido por um compilador que entendeu o elevado intento do programa;
- eram para ser implementadas em lógica randômica mais eficazmente que a seqüência equivalente de instruções de software.

O sistema 801 resultante apóia-se na natureza firmemente integrada de seu hardware e software para obter seu alto desempenho. A implementação do hardware, conduzido pelo desejo de concisão, caracterizou o controle por hardware e instruções que executassem em um único ciclo. Todas as referências à memória foram restritas a leituras e escritas e foram utilizados caches separados para instruções e dados para permitir acessos simultâneos a código e operandos. A CPU do 801 tem 32 registradores de 32 bits de uso geral, e todas as operações são de registrador a registrador. O compilador usa várias estratégias de otimização, incluindo um poderoso esquema de alocação de registradores e otimização global. O compilador assume também a responsabilidade por todas as verificações de referências e proteção no sistema.

Algumas das idéias básicas do projeto 801 alcançaram a costa oeste dos Estados Unidos em meados dos anos 70. Na Universidade da Califórnia, em Berkeley, essas idéias evoluíram em uma série de projetos RISC conduzidos em curso de graduação que produziram o RISC I, RISC II e o SOAR, assim como numerosas ferramentas CAD que facilitaram esses projetos. Esses cursos deram um alicerce para o empenho em estudos sobre avaliação de desempenho, CAD e implementação de computadores.

Como o 801, o processador do RISC I[24] é uma máquina de leitura/escrita controlada por hardware (isto é, os dados podem ser operados somente em registradores e apenas as operações de leitura e escrita acessam a memória), a qual executa a maioria de suas instruções em um único ciclo. Cada uma das 31 instruções usa uma palavra de 32 bits e tem praticamente o mesmo formato. Uma característica especial do RISC I é seu grande número de registradores, mais de cem, que são usados para formar uma série de conjuntos de vários registradores de sobreposição (Overllaping Multiple Register Sets – MRS). Essa característica é usada para fazer chamadas a procedimentos no RISC I mais baratas em termos de tráfego no barramento memória-processador.

É razoável esperar-se que os MRS ofereçam benefícios de desempenho, uma vez que as linguagens de alto nível baseadas em procedimentos (HLL – High Level Language) usam tipicamente registradores para enviar informações específicas para um procedimento.

Quando uma chamada de procedimento ocorre, essa informação deve ser salva, geralmente em uma pilha na memória, e restaurada no retorno do procedimento. Essas operações costumam consumir muito tempo, em virtude de requisições de transferência de dados intrínseca. O RISC I usa os conjuntos de múltiplos registradores para diminuir a freqüência de salvamento e restauração. Ele também aproveita a vantagem de sobreposição entre os conjuntos de registradores para passagem de parâmetros, reduzindo ainda mais a escrita e a leitura na memória, se comparado a esquemas de passagem de parâmetros que usam memória como meio (por exemplo, via pilha)[14].

O arquivo de registradores do RISC I tem 138 registradores de 32 bits organizados em 8 "janelas" de sobreposição (veja Figura 2.1). Em cada janela, seis registradores se sobrepõem com a janela anterior (para a entrada de parâmetros e saída de resultados), e seis registradores se sobrepõem com a janela seguinte (para saída de parâmetros e entrada de resultados). Durante qualquer procedimento, somente uma dessas janelas é realmente acessível. Uma chamada de procedimento troca a janela corrente pela próxima janela incrementando um ponteiro, e os seis registradores de parâmetros de saída tornam-se parâmetros de entrada do procedimento chamado. Similarmente, um retorno de procedimento comuta para a janela anterior, e os registradores de saída de resultados tornam-se registradores de entrada de resultados do procedimento chamador. Assumindo que seis registradores de 32 bits são suficientes para conter os parâmetros, então uma chamada de procedimento não envolve movimento de informação (somente o ponteiro de janela é ajustado). Note-se que os recursos finitos da pastilha limitam os salvamentos por causa de estouros das janelas de registradores[24]. O arquivo de registradores também possui dez registradores globais que são sempre acessíveis.

Os artigos de Berkeley, descrevendo os processadores dos RISC I e II, afirmam que as decisões adotadas produziram grandes aumentos de desempenho em relação às máquinas CISC, como o VAX e o 68000[24,28]. Podem ser feitas algumas reservas sobre esses resultados e os métodos usados para obtê-los. A primeira delas é que os efeitos sobre o desempenho do conjunto reduzido de instruções não foram dissociados dos das janelas de registradores sobrepostos, uma característica que pode ser incorporada em qualquer máquina baseada em registradores de uso geral. Desde que esses fatores de desempenho não foram avaliados independentemente, as pretensões de desempenho em virtude da natureza reduzida dessas máquinas são inconclusivas. Esse ponto será explanado na Seção 2.6.

Pouco depois do primeiro trabalho da Berkeley sobre o RISC I, um processador chamado MIPS (Microprocessor without Interlocked Pipe Stages)[17] surgiu em Standford. O objetivo principal do MIPS é o aumento de desempenho na execução de código compilado. O princípio básico utilizado para alcançar esse objetivo foi expor o paralelismo interno da CPU ao controle do software. Em vez de prover uma eficiente codificação de um

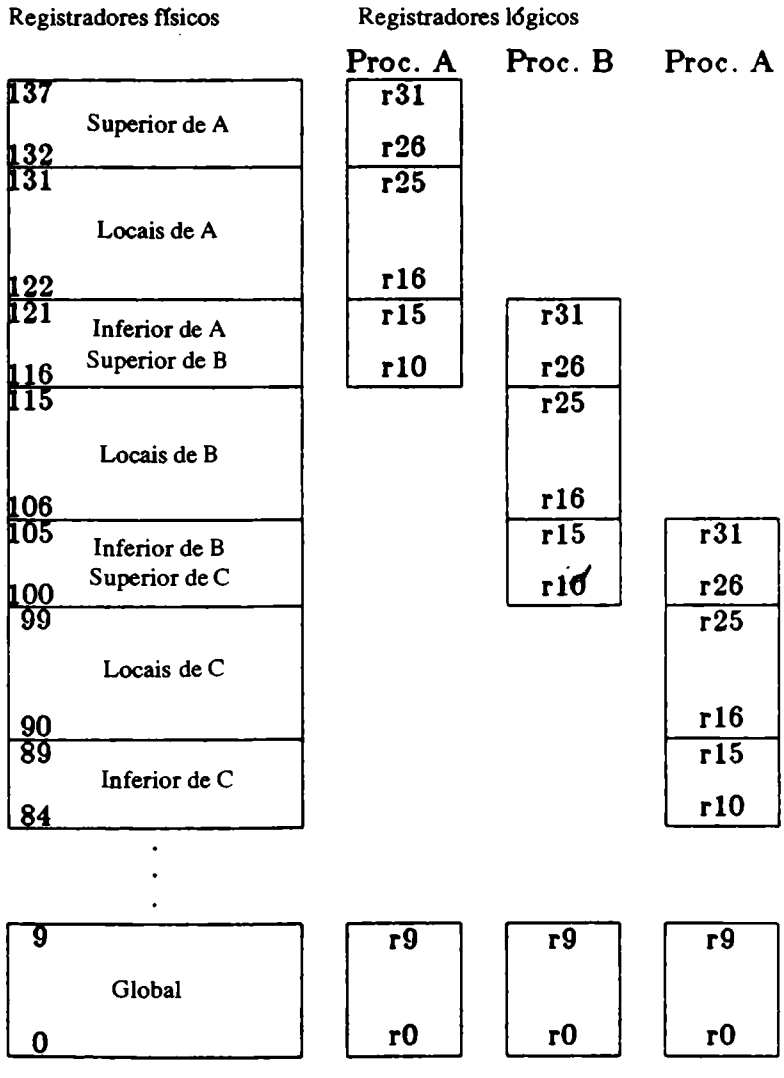


Figura 2.1 Conjunto de registradores sobrepostos do RISC I.

conjunto de instruções simples (como os projetistas do RISC I fizeram), o MIPS apresenta um microengenho com um mínimo de interpretação para o usuário. O processador resultante é um hardware de escrita e leitura cujas instruções assemelham-se a microcódigo (veja Tabela 2.1).

Uma única conseqüência em expor o paralelismo do microengenho ao nível de conjunto de instruções é que, embora o processador do MIPS seja canalizado a recursos, ele não possui hardware para gerenciar e proteger as dependências de recursos e dados no

canal. Essa função deve ser realizada por software (isto é, em tempo de compilação). Tal estratégia é muito diferente de técnicas usadas em outras máquinas com uso extensivo de canais, como no IBM System/360 Model 91, no qual a proteção de recursos entre estágios é feita em tempo de execução por um hardware especial[1]. Transladando essa função de tempo de execução para tempo de compilação, os projetistas do MIPS pretendem produzir uma máquina de computação mais eficiente em reduzido período de tempo.

Tabela 2.1 Conjunto de Instruções do MIPS

Operação	*Operandos*	*Comentários*
Add	src1, src2, dest	Soma de inteiros
And	src1, src2, dest	Operação E Lógica
Ic	src1, src2, dest	Insira byte
Or	src1, src2, dest	Operação OU Lógica
Rlc	src1, src2, src3, dest	Rotação combinada
Rol	src1, src2, dest	Rotação
Sll	src1, src2, dest	Deslocamento lógico à esquerda
Sra	src1, src2, dest	Deslocamento aritmético à direita
Srl	src1, src2, dest	Deslocamento lógico à direita
Sub	src1, src2, dest	Subtração inteira
Subr	src1, src2, dest	Subtração inteira reversa
Xc	src1, src2, dest	Extraia byte
Xor	src1, src2, dest	OU exclusivo lógico
Ld	A[src],dst	Carregue com base
Ld	[src1 + src2],dst	Carga com base indexada
Ld	[src1 >> src2],dst	Carga com base deslocada
Ld	A,dst	Carga direta
Ld	I,dst	Carga imediata
Mov	stc,dst	Carga imediata
St	src1,A[src]	Armazena com base
St	src1,[src2 + src3]	Armazena com base indexada
St	src1,[src2 >> src3]	Armazena com base deslocada
St	src,A	Armazena direto
Bra	dst	Desvio relativo incondicional
Bra	Cond,src1,src2,dst	Desvio condicional
Jmp	dst	Desvio incondicional direto
Jmp	A[src]	Desvio incondicional com base
Jmp	(A[src])	Desvio incondicional indireto
Trap	Cond,src1,src2	Instrução trap
SavePC	A	Salve PC multiestágio após trap ou interrupção
Set	Cond,src,dst	Assinalamento condicional

Vários outros aspectos de projeto para o processador MIPS refletem o desejo de uma mínima complexidade de hardware e alto desempenho. Duas interfaces para memória são fornecidas, uma para instruções e outra para dados. Isto dobra a banda de passagem máxima que apenas uma interface de memória permitiria (um fator importante para o MIPS, já que seu microengenho rápido pararia se não houvesse uma banda de passagem de memória rápida o suficiente). O MIPS é também uma máquina com endereçamento por palavra. Apenas um tamanho de instrução é fornecido no MIPS, e todas as instruções têm o mesmo tempo de execução. Isto, associado à ausência de códigos condicionais, simplifica a manipulação de interrupções e falha de páginas. Qualquer instrução do MIPS que possa gerar falha de paginação garantidamente não modificará o conteúdo da memória antes que a falta de página seja detectada, eliminando a necessidade de salvamento do estado interno para reinício da instrução.

Essas três máquinas, 801, RISC I e MIPS, formam o núcleo das máquinas de pesquisa RISC. Os trabalhos citados descrevem seus princípios detalhadamente, mas algumas de suas características serão referenciadas durante esse artigo. Apesar de outras máquinas RISC serem consideradas e exploradas na indústria e nos meios acadêmicos, a discussão aqui se concentra nessas três.

Nenhuma definição compreensiva sobre RISC emergiu ainda, mas uma é necessária. São propostos aqui seis elementos sinergéticos que parecem essenciais para a filosofia RISC:

1. *Operações de ciclo único.* Facilita a execução rápida de funções simples que predominam no fluxo de instruções de um computador e favorecem um baixo custo interpretativo.

2. *Projeto de leitura/escrita.* Advém da necessidade de operação em ciclo único.

3. *Controle por hardware.* Para produzir operações de ciclo único rápidas. Microcódigo conduz a controle lento e elevação de custo na interpretação.

4. *Relativamente poucas instruções e modos de endereçamento.* Isto facilita interpretação veloz e simples pela máquina de controle.

5. *Formato de instrução simples.* O uso consistente de formato simples facilita a decodificação de instruções por hardware, o que, novamente, aumenta a velocidade do fluxo de controle.

6. *Mais empenho em tempo de compilação.* As máquinas RISC são especializadas em rodar apenas códigos compilados. Isto oferece a oportunidade de mover complexidades estáticas de tempo de execução para o compilador. Um bom exemplo disso é o reorganizador de canal por software usado no MIPS[18].

A lista de características RISC enumerada acima será usada neste capítulo para suprimir afirmativas equivocadas assim como prover um alicerce para debates. Embora alguns aspectos dessa lista possam ser discutíveis, eles servirão, no mínimo, como uma definição de trabalho neste capítulo, para que alguns resultados e implicações do RISC possam ser explorados.

2.3 DESAFIOS DO RISC

Como mencionado anteriormente, as idéias RISC representam um sério desafio para alguns estilos de projeto em uso que têm guiado os projetos de computadores há anos. Esta seção irá examinar cada uma das características definidas anteriormente sob essa ótica. Cada característica será explorada para mostrar como elas diferem de tradições passadas e como elas podem contribuir para tradições futuras.

A arquitetura de leitura/escrita não é um conceito novo, mas certamente não é *a razão de ser* que é para os projetos RISC. Os projetos de leitura/escrita fornecem um nível mais baixo de atomicidade de instrução que as fornecidas por arquiteturas que podem ter acesso múltiplo à memória em uma mesma instrução. Enquanto esse baixo nível de atomicidade requer mais instruções, ele pode contribuir para implementações mais fáceis. Isto é verdade para máquinas que têm de suportar memória virtual e, portanto, têm de recuperar erros de acesso de memória. Criar uma máquina que possa recuperar tal erro e reiniciar coerentemente é direcionar-se por uma máquina RISC. Torna-se mais difícil ainda quando podem ser feitos acessos múltiplos à memória em uma mesma instrução.

É difícil argumentar contra o desejo de controle por hardware. O tempo de ciclo mais rápido e a menor área de controle que deriva de controle por hardware são sempre atrativos. Isto combina razoavelmente com a filosofia RISC, que defende uma máquina suficientemente simples para que o controle por hardware seja possível. Para máquinas maiores, o controle por hardware pode se tornar um pesadelo. A regra para os projetistas é "Instruções RISC devem rodar na velocidade RISC", significando que o subconjunto de instruções comumente usado deve rodar como se fosse controlado por hardware. Obviamente, o único modo de fazê-lo é controlá-los por hardware, tendo-se duas alternativas. Uma é projetar um RISC. A outra requer a questão: "Sistemas controlados por hardware e microcódigo podem ser efetivamente unidos?" Se há exemplos comerciais de tais máquinas não se está ciente disso.

Tendo-se relativamente poucas instruções e modos de endereçamento em uma arquitetura, pode-se ter dois efeitos positivos na implementação. Primeiro, isto facilita um projeto rápido. Projetar qualquer processador comercial é, por natureza, uma tarefa complexa; com toda a documentação necessária e testes, um processador pequeno e simples

será sempre mais fácil de projetar que um maior e mais complexo. O segundo efeito positivo trata de recursos como espaço para a pastilha, potencial disponível e espaço de placa. Um processador simples necessitará de menos recursos para implementação. Isto poderá deixar recursos para características que não são da arquitetura, como caches, tradução de endereços, E/S em pastilha (em placa). Isto fornece uma flexibilidade que não se encontra em projetos de processadores mais complicados.

Nem todos os pontos dessa definição de RISC podem ser aplicados somente para processadores pequenos e simples. Por exemplo, o RISC não é o único estilo de arquitetura que pode ser útil para formatos de instrução simplificados. Os implementadores do CISC já sentiram o pânico de prover instruções complexas, de tamanho variável, e codificação vertical. As arquiteturas como a do VAX têm um tendão de Aquiles limitando o desempenho em seus modos de endereçamento emaranhados e instruções de múltiplos operandos. Decodificar tais seqüências de instruções é, por natureza, ou vagaroso ou caro ou os dois. Não será surpresa se arquiteturas futuras, complexas e massivas, aparecerem com formatos de instruções mais claros dos que existem hoje. Algumas grandes arquiteturas recentes, como a série Ridge 32/100, demonstra essa tendência.

A ênfase em tecnologia de compiladores que os projetos RISC têm mostrado também passa a ter um efeito positivo em todos os projetos de computadores. Enquanto escrever código assembly para o 360 ainda for predominante, um código compilado executável eficiente é um objetivo apropriado para qualquer arquitetura nova. Além disso, a noção de mover funcionalidades do tempo de execução para o tempo de compilação finalmente teve a atenção apropriada no mundo RISC. Deveria ser dado o mesmo nível de atenção a esse princípio em todos os projetos de computador.

A vantagem de operações de ciclo único é uma característica RISC que, embora intuitivamente apareça em muitos pontos, não tem sido demonstrada. Infelizmente, projeto de computador não é uma ciência; nenhum teorema pode ser provado demonstrando a superioridade de projetos com ciclo único. Enquanto é fácil acreditar que máquinas de ciclo único tenham vantagem sobre máquinas tradicionais de microcódigo, há pouca exploração de esquemas híbridos que possam misturar operações de único e múltiplos ciclos.

2.4 RISC COMERCIAIS

Até agora a discussão sobre máquinas RISC tem se limitado a projetos de estudo. Nesta seção serão discutidas algumas máquinas comerciais que consideramos com algumas qualidades RISC. Mais especificamente, serão discutidas máquinas cujas especificações têm sido amarradas a implementações do hardware por razões de desempenho. Isto é um atributo-chave da filosofia RISC e é válido discutir máquinas que tiveram decisões de

projeto significantes que atravessam os limites tradicionais de hardware/software. Esta seção será fechada com uma discussão breve sobre várias máquinas que por várias razões têm sido chamadas RISC mas fogem da definição apresentada na última seção.

2.4.1 O INMOS TRANSPUTER

O transputer[20] representa um caminho novo para projetar sistemas de microprocessadores VLSI. Os microprocessadores típicos são projetados para aplicações de uniprocessadores e são padronizados ao nível de instruções. O principal objetivo do Inmos é prover uma família de componentes VLSI programáveis de alto desempenho (transputers) contendo enlaces de comunicação de banda larga para a criação de sistemas multiprocessadores concorrentes. O nível de padronização para transputers é o *occam*, uma linguagem de programação na qual os processos concorrentes comunicam-se via passagem de mensagens através de canais especificados. Os modelos occam de processos e comunicação são suportados diretamente por implementações de transputers.

Padroniza-se a família de transputers no nível occam para permitir ao hardware do transputer englobar futuras tecnologias VLSI e ser otimizado para propostas de processos especiais mantendo-se a compatibilidade com o software básico do occam. Como tal, cada transputer será projetado para fazer o mais eficiente uso da tecnologia disponível, a fim de maximizar o desempenho. É nesse ponto que o transputer começa a ser associado com a filosofia RISC. Os projetos de transputers são abertos para decisões que atravessam os limites convencionais de arquitetura/implementação, hardware/software e tempos de compilação/execução para alcançar alto desempenho tanto quanto os modelos occam de processos e comunicação suportem. Os primeiros resultados dessa abertura podem ser vistos na implementação inicial dos transputers.

As duas maiores características do primeiro membro da família de transputers, o IMST424, são uma CPU com microcódigo simples e 2 kbytes de memória na pastilha. As razões para a escolha de uma memória na pastilha grande e um microengenho simples podem ser explicadas pelos constrangimentos de implementação em silício atuais. A memória na pastilha oferece acessos rápidos para uma larga quantidade de dados sem se utilizar de comunicação com a memória externa à pastilha. A memória é também bem adequada a implementações em VLSI por causa de sua estrutura regular (também os principais produtos da Inmos têm sido memórias semicondutoras). Entretanto, os 2 kbytes de memória na pastilha usam uma parte significante da área de silício limitando os recursos de hardware disponíveis para o microengenho. Isto tem implicações diretas no conjunto de instruções, porque o largo espaço necessário para o microcódigo de um conjunto de instruções complexo não é disponível. Como tal, o conjunto de instruções do T424 tem formato

muito simples, codificado em um byte, e a maioria das instruções roda em ciclo único.

O T424 tem apenas seis registradores (Figura 2.2). Esse número, embora pequeno, não é um grande prejuízo para o desempenho do processador, por causa da memória na pastilha, que fornece acesso aos dados com velocidade próxima à desses registradores. O número reduzido de registradores também reduz a quantidade de estados que necessitam ser salvos e restaurados nas chamadas de procedimentos e trocas de processos. Os seis registradores são:

- os registradores A, B e C, que fazem uma avaliação da pilha. Esses registradores são referidos implicitamente por instruções eliminando a necessidade de bits nas mesmas para especificar registradores, simplificando o formato das instruções;
- o ponteiro da área de trabalho aponta para a área na memória onde residem as variáveis locais;
- o ponteiro para próxima instrução;
- o registrador de operando que é usado para formar operandos das instruções.

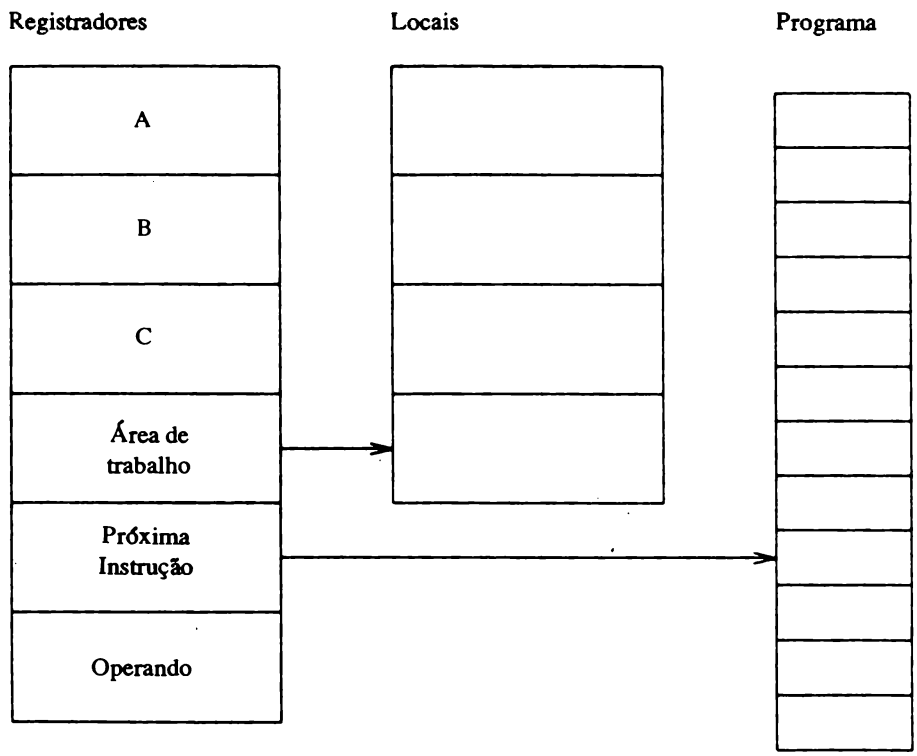


Figura 2.2 Registradores do T424

O conjunto de instruções do T424 tem apenas um formato de instrução. Todas as instruções são de um byte, que é dividido em duas partes: um campo de função de 4 bits e um campo de dado de 4 bits (Figura 2.3). As instruções são divididas em três grupos:

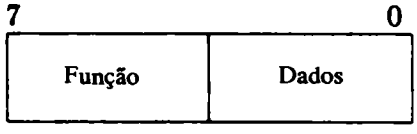


Figura 2.3 Formato de instrução do T424

- *Funções diretas.* Esses 13 opcodes constroem a maioria das operações comuns usadas em um programa, como a carga e armazenamento de variáveis locais da área de trabalho, carga e soma de constantes, saltos e chamadas de procedimentos;

- *Funções de prefixo.* Desde que apenas 4 bits de dados podem ser expressados em uma instrução, é necessário um cache para operandos mais longos. Esta é a função de duas instruções de *prefixo, prefix* e *prefix negativo.* A instrução *prefix* carrega os 4 bits de dados nos 4 bits menos significativos (low order) do registrador de operandos e, então, desloca o conteúdo em 4 bits. O *prefix negativo* opera similarmente, exceto que o conteúdo do registrador de operando é complementado antes do deslocamento. Assim, operandos de qualquer tamanho (desde que não ultrapassem o tamanho do registrador de operando) podem ser construídos.

- *Funções indiretas.* O opcode *operate* faz com que seus operandos sejam interpretados como uma operação a ser executada com os valores na pilha de avaliação. Um exemplo dessa operação seria somar os conteúdos dos registradores A e B, armazenar o resultado em A e copiar o conteúdo de C em B. Essa instrução permite especificar, com 1 byte, 16 operações na pilha de avaliação. É possível também especificar qualquer número de operações por meio do uso das instruções de prefixação.

O T424 também suporta o modelo occam de concorrência. O processador tem um escalonador que o compartilha entre os processos ativos. Esses processos são mantidos em duas listas ligadas de prioridades diferentes. Os registradores em hardware são usados para apontar o primeiro e o último processo da lista. Os processos novos são alocados no fim da lista, e, quando um processo não está habilitado para ser processado, seu ponteiro de instrução é salvo na sua área de trabalho, e o próximo processo na lista será processado.

O modelo de comunicação occam é implementado no T424 também. As mensagens são passadas através de canais que podem ser internos ao processador ou conectados entre dois processadores. Os canais internos são implementados no T424 usando uma palavra na memória, e os canais externos são construídos usando os enlaces de comunicação

ponto a ponto. A mesma seqüência de código pode ser usada para os canais externos e internos, permitindo que os processos sejam escritos sem o conhecimento da alocação real dos processos. Dessa forma, um programa occam pode rodar qualquer configuração de transputers; somente informações sobre detalhes de configuração do sistema precisam ser mudadas.

A Inmos tem reportado quadros de desempenho impressionantes para o T424. Colin Whitby-Stevens declara que para a seqüência de instruções típicas comumente usadas pode ter uma taxa de execução de 15 MIPS. As taxas de MIPS podem ser sempre encaradas como suspeitas, especialmente quando usadas como medida de comparação entre máquinas diferentes. Deve-se conhecer o tipo de instruções que são executadas (operações, tipo de dado e modos de endereçamento) e onde tais instruções e os dados que elas manipulam são encontrados (por exemplo, em um cache, fila de pré-aquisição, ou memória principal), para entender-se exatamente que tipo de desempenho está sendo indicado. Isto é especialmente verdadeiro para a taxa de 15 MIPS para o T424. Esta é a taxa máxima para instruções de ciclo único em memória na pastilha. A degradação do desempenho da memória externa (um caso provável desde que apenas 2 kbytes de memória estão na pastilha) e instruções mais vagarosas (por exemplo, troca de contextos e instruções de multiplicação) não são incorporadas nessas considerações. Deve-se lembrar também que as instruções típicas do T424 executam somente operações simples, ao passo que, em uma máquina mais complexa, pode-se acoplar em uma instrução o que requereria toda uma seqüência de instruções no T424.

A memória na pastilha do T424 provê acesso rápido a instruções e dados. Entretanto, o conteúdo da memória deve ser manipulado explicitamente pelo programador. A gerência dessa memória não é transparente para o programador como um cache usualmente o é. Assim sendo, quando é necessário mais de 2 kbytes de memória o programador deve alocar cuidadosamente a memória da pastilha para os processos mais críticos do recurso tempo, para obter alto desempenho.

Deve-se notar que, embora occam e transputer possam ser ferramentas efetivas para a construção de sistemas altamente concorrentes, eles não provêem uma solução geral para decompor um problema em partes concorrentes ou planejar a interação de suas partes. O projetista do sistema deve tomar decisões relativas a se os processos são mais efetivos quando operando concorrentemente, como os processos concorrentes poderiam ser distribuídos nos transputers, e quando os processos se comunicariam entre si. Estes são os mesmos problemas que têm ocorrido em projetos de computadores paralelos há anos. Norman Aleizer, consultor da Arthur D. Little Inc., entrevistado na *Electronics Week*, disse: "Exceto para processamento de sinal puro, o mercado de processamento paralelo tem uma história de falhas"[21]. Isto não quer dizer que a família do transputer fra-

cassou, mas observa-se que occam e transputer não ofereceram uma solução total de sistema. Várias decisões de projeto não triviais ainda precisam ser feitas para compreender-se efetivamente um sistema concorrente usando transputers.

2.4.2 ACORN RISC MACHINE (ARM)

A Acorn Computers de Cambridge, Inglaterra, anunciou o desenvolvimento do Acorn Risc Machine (ARM), um processador VLSI de 32 bits[2]. O ARM foi projetado em um período de dezoito meses e todos os objetivos de projeto foram alcançados na primeira pastilha. O ARM propõe exceder o desempenho de microprocessadores comerciais disponíveis, como o Motorola 68000 e o National Semiconductor 32016.

O ARM é a resposta do objetivo da Acorn de construir computadores pessoais de baixo custo e estações de trabalho usando tecnologia de 16 ou 32 bits. Os microprocessadores de 16 e 32 bits existentes foram avaliados, e a Acorn concluiu que eles não poderiam ser incorporados em computadores pessoais de baixo custo sem um compromisso significante com o desempenho do sistema. A Acorn menciona as longas operações não-interruptíveis nos microprocessadores existentes, que podem aumentar o tempo de resposta a interrupção, como um fator limitante do desempenho do sistema, e concluiu que os microprocessadores existentes podiam não suportar requisições de E/S intensa sem hardware de DMA caras.

Desde que os microprocessadores existentes foram avaliados como impróprios para máquinas de baixo custo e alto desempenho, a Acorn começou a considerar a possibilidade de projetar seu próprio microprocessador. Alguns critérios de projeto considerados por eles foram:

1. A latência de interrupções do processador deveria ser reduzida, por causa das razões citadas acima.
2. O desempenho da nova máquina deveria ser no mínimo comparável ao das máquinas comerciais existentes.
3. O novo microprocessador deveria suportar sistemas de memória virtual, implicando que todas as instruções precisariam ser reiniciáveis.
4. A tarefa de projetar deveria ser simples, considerando os recursos disponíveis para a Acorn.

Primeiramente foi adotada a filosofia RISC para chegar ao quarto critério. Entretanto, uma máquina simples pode melhorar a latência de interrupções e simplificar o

hardware necessário para permitir que as instruções sejam reiniciáveis (critérios 1 e 3).

Algumas características importantes do ARM são:

- o ARM tem um barramento de dados de 32 bits e um de endereços de 26 bits. A memória é enderaçada por byte. Todas as instruções são de 32 bits e alinhadas em palavras. O ARM tem 16 registradores de uso geral, dos quais dois são usados como contador de programa e como registrador indicador de retorno de sub-rotina. Todas as operações são feitas em palavras de 32 bits. As instruções de armazenamento e carga são usadas para conversão entre byte e palavras;

- todas as operações lógicas e aritméticas são feitas em registradores. Essas instruções especificam um registrador de origem, um segundo registrador ou uma constante de 8 bits, e um terceiro registrador de destino. O segundo operando-fonte pode ser trabalhado (deslocado/circulado) antes de ser usado como operando;

- as operações de carga e armazenamento podem usar uma base geral acrescidas de indexação, ou uma base mais uma constante de 12 bits para endereçamento da memória. Qualquer registrador pode ser usado como uma base, e o registrador de indexação pode ser deslocado antes de somado à base. O registrador base pode ser atualizado ou não, antes ou depois da transferência, conforme desejado. Instruções para transferência de múltiplos registradores de/para posições contíguas de memória também estão disponíveis;

- instruções de desvio utilizam alinhamento em palavra, deslocamento de 24 bits que permite desvios para qualquer lugar dentro da memória endereçável. As instruções de desvio podem especificar que o contador de programa seja salvo no registrador indicador de chamadas de procedimento, para implementar chamadas e retornos de procedimentos;

- os modos supervisor e de atendimento de interrupção têm acesso a nove registradores não disponíveis para o uso de programas de usuário. Esses registradores são usados para minimizar a latência de interrupções e trocas de contexto, como também para prover suporte para simular um canal DMA;

- reiniciação de instruções após uma exceção não é manipulada completamente pelo hardware. O critério utilizado pelo ARM foi o de que seria possível restaurar o contexto do processador. Como tal, a única garantia é que todas as informações necessárias para o restabelecimento do estado do processador depois de um aborto são preservadas. O software deve guardar e restabelecer o estado apropriado antes de uma instrução ser executada novamente;

- o processador ARM suporta diretamente os ciclos de modos de paginação para DRAM.

Isto pode produzir uma banda de memória mais elevada do que não usar ciclos de modos de paginação.

2.4.3 O JARGÃO RISC

O termo RISC tem se tornado popular no meio técnico e tem, por extensão, transformado-se em jargão. Várias companhias de computadores estão desenvolvendo sistemas de computadores novos que eles afirmam ser RISC, mas que possuem características muito diferentes das descritas acima.

Como exemplo de companhias que se consideram ter máquinas RISC, considere duas companhias que declaram ter criado o primeiro computador comercial RISC: a Ridge Computers e a Pyramid Technology. Ambas as máquinas têm formato de instrução restrito, uma característica que elas compartilham com máquinas RISC. Mas a máquina da Pyramid não é de leitura/escrita, e ambas as máquinas, Ridge e Pyramid, têm instruções de tamanho variado, envolvem interpretação de ciclo múltiplo, e empregam microcódigo. Além disso, embora seus conjuntos de instruções possam ser reduzidos em comparação ao VAX, a Pyramid tem quase 90 instruções e o Ridge tem mais de cem. O uso de microcódigo nessas máquinas não é inesperado, por razões de custo/desempenho. A máquina da Pyramid também tem um sistema de conjuntos múltiplo de registradores que deriva do RISC I da Berkeley, mas, como será discutido mais tarde, essa característica independe da filosofia de projeto RISC. Sob o ponto de vista do marketing e pela tecnologia essas máquinas podem ser de sucesso, mas não são RISC.

2.5 PONTOS DE CONFUSÃO SOBRE RISC/CISC

Há duas noções erradas prevalentes sobre RISC e CISC. A primeira é sobre a controvérsia RISC e CISC que parece implicar que o domínio da discussão seria restrito à seleção de candidatos a um conjunto de instruções. Apesar da especificação de instruções, seus formatos e o número das mesmas ser o foco básico da maior parte da literatura RISC, uma melhor generalização da filosofia RISC vai bem além dessa atividade reduzida. Mais propriamente, ela conota uma disposição concisa e independente para tomar decisões de projeto sobre arquitetura/implementação, hardware/software e limites entre tempo de compilação/execução visando maximizar o desempenho (como avaliado em contextos específicos).

A controvérsia RISC/CISC também parece implicar que qualquer máquina apresentada pode ser classificada tanto como CISC ou RISC, e que a tarefa primária de-

frontada por um projetista está em escolher o estilo de projeto mais adequado para uma aplicação específica. Mas a classificação RISC/CISC não é uma dicotomia. Os RISC e os CISC estão em ângulos diferentes de um espaço de projeto multidimensional e contínuo. O que é necessário não é um algoritmo pelo qual pode-se escolher entre RISC e CISC; ao contrário, o objetivo deveria ser a formulação de um conjunto de técnicas extraídas da experiência em CISC e dos princípios de RISC que possa ser utilizado por um projetista para associar apropriadamente funcionalidade ao nível de implementação, dentro de um sistema de computador[9,15,16].

Infelizmente, como foi colocado[7], os termos *reduzido* e *complexo* têm sido contrapostos em discussões da filosofia. De fato, duas dimensões ortogonais de conjuntos de instruções estão em questão aqui: tamanho (reduzido *versus* grande) e complexidade (simples *versus* complexo). A primeira dimensão concerne ao número de instruções (modos de endereçamento, número de valores possíveis nos campos de instruções em geral) que caracteriza uma arquitetura. O outro concerne à complexidade funcional das instruções que poderia ser representado pelo número de operações "primitivas" necessárias para sintetizá-las. Essa dimensão é mais difícil de medir, pois pode haver mistura entre instruções simples e complexas dentro da mesma arquitetura de computador.

É verdade que "reduzido" e "simples" têm uma forte ligação dentro do contexto de projeto RISC, assim como "grande" e "complexo" normalmente são do domínio CISC. Isso não precisa ser uma norma. Simplicidade tem um significado diferente para projetistas de pastilhas, arquitetos de computadores e outras pessoas envolvidas no projeto. O VAX tem sido apontado como sendo de arquitetura complexa. Entretanto, pela ótica de seus projetistas, o VAX teria um conjunto de instruções grande mas simples. A definição de simplicidade usada neste contexto foi:

- aqueles atributos (outros além de preço) que tornaram o sistema de minicomputadores atrativo;
- estes incluem compreensibilidade, compatibilidade e facilidade de uso[28].

É questionável se esse objetivo foi alcançado ou não, especialmente porque os itens que são maciços tendem a ser complexos. Será discutido adiante que de certa forma o termo simplicidade pode ser de importância primordial.

No que se refere a suporte, tendo-se apenas operações simples, os proponentes do RISC freqüentemente observam os efeitos nocivos do uso de instruções complexas. Mesmo assim, a popularidade em incorporar funções especializadas como comunicação interprocessos (IPC – Interprocess Communication) não parece diminuir. Os projetistas do EXLS/6400 declaram[29], por exemplo:

"Uma característica de arquitetura chave que permite ao sistema operacional lutar contra a grande variabilidade de seu ambiente de hardware é o microcódigo e o sistema de mensagem implementados em hardware. O uso de mensagens nos permite fazer opções na arquitetura da CPU e do sistema operacional que aumentam a eficiência de processadores adicionais".

Mas concordamos com uma das críticas do RISC em relatórios publicados sobre essas máquinas: não é suficiente mostrar que uma instrução complexa executa mais rapidamente que uma seqüência equivalente de instruções primitivas. Também deve-se mostrar que o efeito seja o aperfeiçoamento do desempenho do sistema. Acreditamos que esse aspecto do problema deve ser uma preocupação no projeto.

Há vários outros ambientes de computação, tal como tempo real ou sistemas de processamento de sinal, onde seria difícil argumentar contra funções complexas de suporte na arquitetura e implementação. Mais genericamente, Radin escreveu[27]:

"Geralmente é verdade que implementar uma função complexa em lógica randômica resulta que sua execução seja significativamente mais rápida que programá-la como uma seqüência de instruções primitivas. Como exemplos temos a aritmética em ponto flutuante e a multiplicação em ponto fixo. Não temos objeção a essa estratégia, desde que a freqüência de uso justifique o custo e, mais importante, desde que essas instruções complexas de forma alguma diminuam a velocidade das instruções primitivas".

Confirmamos essa declaração, mas argumentamos que a freqüência de uso não é um critério suficiente para justificar uma dada instrução. Como Clark e Levy disseram[6]:

"Agregação de estatísticas separadas não podem guiar o projeto de um conjunto de instruções planejado para linguagens e aplicações diferentes. Em particular, as instruções que não são usadas freqüentemente podem ser críticas para alguns possíveis usuários".

A noção de que funções complexas diminuem a velocidade de ações simples de um computador parece ser um problema real que nos impede de ter o melhor. Acreditamos que um estudo sério nas áreas de partição funcional, interpretação de instruções e decodificação distribuída produzirá estruturas de computador que reduzirão ou eliminarão esse efeito. Até que os estudos sejam diretamente dirigidos para esse problema, uma compreensão grande das verdades científicas e os princípios envolvidos, em oposição ao folclore correntemente disseminado, não é possível.

Uma conseqüência da atitude "nós ou eles", evidenciada na maioria das publicações RISC, é que o desempenho reportado de uma máquina particular (por exemplo, RISC I) pode se tornar muito difícil de interpretar se as vantagens advindas das diversas

decisões de projeção não forem apresentadas individualmente. Um projetista defrontando-se com uma grande gama de alternativas precisa de um suporte mais específico do que uma medida de desempenho monolítica como tudo ou nada.

Um exemplo de como o tema de campo de ação pode estar confuso é encontrado em um artigo recente[3]. Criando-se uma máquina com apenas uma instrução, seus autores afirmam ter delimitado o espaço de projeto RISC, com sua máquina em um extremo desse espaço e o RISC I (com 31 instruções) no outro extremo. Mas um número absoluto de instruções não pode ser o critério exclusivo para classificar uma arquitetura como RISC ou CISC. Esse modelo está longe de ser proveitoso: ele ignora aspectos de modos de endereçamento e a complexidade associada; ele falha no manejo do par compilador/arquitetura; ele não fornece meios para avaliar a implementação de outras decisões de projeto além do conjunto de instruções como conjuntos de registradores, caches, gerenciamento de memória, operações de ponto flutuante e co-processadores.

Outro engano propagado por esse artigo é que todo sistema é composto por hardware, software, código e aplicação. Isso infelizmente desconsidera o sistema operacional e o overhead (tempo gasto pela máquina em funções internas); a necessidade de sistema operacional não pode ser ignorada na maioria dos sistemas. Essa área tem tido pouca atenção dentro dos estudos RISC (em contraste a vários esforços feitos para o CISC)[4,6].

Uma consideração para projetos de sistemas de computadores que recebeu muita atenção nos primeiros argumentos a favor do RISC foi a de que projetos mais simples podiam ser concretizados mais rapidamente dando-lhes uma vantagem de desempenho sob máquinas mais complexas. Juntando-se as vantagens econômicas de "pegar o mercado primeiro", isto foi suposto permitir-se implementações de tecnologia relativamente antigas. Dentro desse contexto, o VAX de 32 bits da DEC é muito mais interessante.

O VAX qualifica-se como um CISC. De acordo com artigos publicados, o MicroVAX-32, uma implementação VLSI de preponderância do conjunto de instruções do VAX, foi projetado, concretizado e testado em um período de vários meses. Pode-se especular que esse pequeno período de gestação foi possível, em grande parte, em virtude da considerável sabedoria da DEC em implementar a arquitetura do VAX (produtos existentes, incluindo o 11/780, 11/730, 11/750 e o VLSI-VAX). Isso não seria possível se a DEC não tivesse, primeiro, criado um conjunto de instruções padronizado. Mas padronizar a esse nível é precisamente o que a filosofia RISC é contrária. Tal padrão restringe o compromisso de hardware/software não-convencional que os RISC investem. Sob o ponto de vista comercial é significante que o MicroVAX-32 tenha nascido em um mundo onde os assemblers compatíveis, compiladores e sistemas operacionais abundem, o que

o que certamente não é o caso para um projeto RISC, onde toda nova geração deve começar do zero.

Tais problemas com projetos de sistemas RISC podem incentivar projetistas de RISC comerciais a definir novos níveis de padronização para obter algumas vantagens de múltiplas implementações suportando uma interface única. Uma opção possível para tal interface pode ser definida como uma linguagem intermediária, como alvo para todos os compiladores. A linguagem intermediária poderia ser transladada para um código de máquina ótimo para cada implementação. Uma forma simples desse processo de translação seria por meio de escalonamento de recursos em um nível baixo (por exemplo, gerenciamento canalizado e alocação de registradores).

Pode-se notar que o MicroVAX-32 não implementa diretamente toda a arquitetura do VAX. Têm sido feitas insinuações de que isso de certa forma suportaria a tendência RISC para emulação de funções complexas por software[28]. Essas insinuações não são razoáveis. O VAX não se aproxima dessa definição do RISC violando todos os seis critérios expostos. Para começar, qualquer VAX, por definição, tem tamanho de instrução variável e não é máquina de carga/armazenamento. Além disso, o MicroVAX-32 tem execução de instruções com multiciclo, conta com uma máquina de controle em microcódigo e interpreta todo o conjunto de modos de endereçamento VAX. Finalmente, o MicroVAX-32 executa 175 instruções na pastilha, o que não pode ser considerado um número "reduzido" de instruções.

Uma perspectiva melhor do MicroVAX-32 poderia mostrar que há de fato fatores de custo/desempenho onde a implementação por microcódigo para certas funções é imprópria e a implementação por software seria melhor. A importância de se fazer cuidadosamente essa associação de função e nível de implementação (software, microcódigo ou hardware) tem sido amplamente demonstrada em vários artigos sobre RISC. Essa simples preocupação é evidenciada também em várias máquinas CISC. No caso do MicroVAX-32, as instruções de ponto flutuante foram levadas tanto para um co-processador como para rotinas de emulação por software. Os numerosos integrados de processadores de ponto flutuante disponíveis atestam o bom senso contemporâneo dessa separação. Também são emuladas as instruções de console, de decimal e manipulação de séries de caracteres. Desde que várias dessas instruções não são freqüentemente usadas, não são críticas em relação a tempo de execução, ou não são geradas por vários compiladores, seria difícil considerar essa decisão de projeto ruim, para produzir um VAX mais barato. O MicroVAX-32 também mostra que é possível que projetistas de computadores inteligentes e competentes, que tenham a noção correta de mapeamento de função para nível, achem a microcodificação uma técnica apreciável. Trabalhos já publicados sobre RISC não permitem essa possibilidade.

Na medida em que a tecnologia VLSI se desenvolve, o nível de integração irá aumentar, até que se seja capaz de incluir a memória principal e seu gerenciamento na mesma pastilha com a CPU. Uma vez que esse avanço de tecnologia seja alcançado, a filosofia RISC não oferecerá opções de como obter vantagens de densidade maior de portas. A filosofia RISC parece estar basicamente em desavença com a idéia de milhões de transistores em uma única pastilha. Isso parece combinar bem com a tecnologia atual (por exemplo, os integrados de GaAs atuais restringem-se entre 10 e 20 mil portas), mas eles se tornarão menos atrativos com o aumento da fabricação.

O ambiente de aplicação é de crucial importância também em projetos de sistemas. O conjunto de instruções do RISC I foi projetado especificamente para rodar a linguagem C eficientemente, e ele parece fazê-lo razoavelmente bem. Nos estudos do RISC I também foi investigado o ambiente de computação SMALLTALK-80. Em vez de avaliar o RISC I como uma máquina SMALLTALK, os estudiosos do RISC I projetaram um novo RISC, e as simulações apresentaram resultados de desempenho encorajadores. Mas projetar um processador para rodar bem uma linguagem é um esforço qualitativamente diferente de criar uma máquina simples, que deva exibir no mínimo um desempenho aceitável para diversas linguagens (por exemplo, o VAX). Embora os estudos RISC ofereçam valiosas visões sobre bases de linguagens, mais ênfase em anomalias interlinguagens, identidades e tradeoffs é urgentemente necessário.

São especialmente equivocadas as declarações RISC que se referem ao tempo em projeto economizado pela criação de uma máquina simples em vez de uma complexa. Tais declarações parecem razoáveis. Entretanto, existem diferenças substanciais entre ambientes de projetos para um projeto acadêmico particular (tal como MIPS ou RISC I) e para uma máquina com o tempo de vida medido em anos, requerendo software substancial e investimento em suporte. Como foi declarado em um artigo da *Electronics Week,* por R. Davior Lowry, gerente de desenvolvimento de mercado da Denelcor: "Note que equipes de desenvolvimento de produtos comerciais geralmente iniciam um projeto pesando impactos de lucros e perdas nas decisões de projetos. Uma universidade não tem de se preocupar com isso; então, freqüentemente, ocorre que vários projetos são iniciados e não terminados... Isto não quer dizer que o valor das pesquisas feitas por eles seja reduzido. Isso significa, no entanto, que é muito difícil alguém reinvestir no sistema para torná-lo um produto comercial"[21]. Para que um produto se torne viável, é necessário fornecer uma grande gama de documentação, treinamento para usuário, facilitar a coordenação entre fabricação e produção, com a previsão de futuras expansões. Esses fatores distorcem tais comparações de tempo de projeto, e comparações dessa natureza devem ser vistas com suspeita.

Na literatura RISC, vê-se uma grande propensão tendenciosa, mas profunda,

referente aos aspectos de um sistema de computador relativos a desempenho. Mais claramente, se todos os outros atributos forem iguais, um melhor desempenho deve ser considerado um aperfeiçoamento para a máquina. Entretanto, consideramos possível atribuir-se tanta importância à dimensão desempenho em um sistema de computador. Desde que o desempenho é a medida mais quantificável de uma máquina, ela é a mais discutida medida – não porque o desempenho seja sempre inerentemente mais valioso que os outros parâmetros, mas porque testes de avaliação são um caminho mais fácil entre as alternativas de comparação de sistemas. É um erro guiar-se pelo desempenho cegamente sem conhecimento explícito do que está sendo tratado por ela.

Mesmo assim, as declarações sobre desempenho nos projetos RISC, talvez a mais interessante de todas as afirmações RISC, não são ambíguas. O desempenho quando medido por estreitos testes de avaliação de baixo nível, como tem sido feito pelos pesquisadores do RISC (por exemplo, solução recursiva do jogo Torre de Hanói) não é a única medida em sistemas de computação, e, em alguns, não é nem mesmo o mais interessante. Para vários computadores modernos, a única figura de desempenho de mérito é o *número de transições por segundo,* o qual não tem correlação direta ou simples com o tempo que ele usa para calcular a função de Ackermann. Enquanto milhões de instruções por segundo podem ter significado em alguns ambientes de computação, recuperação, avaliação e tempo de resposta são muito mais importantes em outros, tal qual sistemas de defesa espacial e aviação. O sistema de detecção de erro incorporado nessas máquinas, em todos os níveis, pode reduzir o período de relógio básico e degradar substancialmente o desempenho. Mas reduzir o desempenho é tolerável, reduzir o tempo pode não ser. Em um extremo, a simples aplicação das regras RISC para projetar um conjunto de instruções resultaria em um computador otimizado para rodar sua tarefa mais comum: diagnósticos. Em termos de freqüência de instruções, naturalmente aplicações de controle de vôo constituem um caso especial trivial e poderia não ser dada muita atenção. Enquanto esse exemplo é humoristicamente implausível, é válido enfatizar que nos esforços para quantificar projetos de sistemas deve-se ter atenção não somente na freqüência de execução de instruções, mas também nos ciclos consumidos por execução de instrução. Levy e Clark fazem esse tipo de referência em relação ao conjunto de instruções do VAX[22], todavia ele ainda tem de aparecer em alguns estudos do RISC relatados.

Para as aplicações onde o desempenho é uma preocupação de primeira ordem, pode-se deparar com a tarefa de quantificá-la. O esforço da Berkeley no RISC I de estabelecer a capacidade de sua máquina é louvável, mas antes de tirar conclusões deve-se examinar cuidadosamente os programas de testes de avaliação usados. Citando um artigo recente:

"As previsões de desempenho (RISC I e RISC II) foram baseadas em programas

pequenos. Esse tamanho pequeno foi ditado pela confiabilidade do simulador e do compilador, pelo tempo disponível para simulação e a inabilidade dos primeiros simuladores para manipular as chamadas do sistema Unix[26]".

Alguns desses programas "pequenos" realmente executam milhões de instruções, embora sejam programas bastante reduzidos em termos do escopo da função. Por exemplo, o programa Torre de Hanói, quando executado no 68000, gasta acima de 90% de seu acesso à memória executando procedimentos de chamada e retorno. Os pesquisadores do RISC I e RISC II recentemente reportaram resultados de um teste de avaliação grande[26], mas a importância de testes de avaliação grandes e não-homogêneos na medição de desempenho ainda é perdida na avaliação de computadores comerciais e acadêmicos, aos quais sucumbiram no errado conceito de que *microtestes de avaliação isolados* representam uma medida útil.

Medir sistemas de computador requer que a carga de processos seja planejada de tal forma que os resultados de medição possam ser interpretados de forma prática. A arte de avaliar envolve prover programas que como um todo representem o processamento carregado visto por uma máquina em uso real.

Entretanto, testes de avaliação são ainda uma arte. Existe uma pequena concordância sobre como caracterizar uma carga de processamento típico, muito menos do que na criação de testes de avaliação que representem aqueles carregamentos com exatidão. Mesmo em testes de avaliação que podem correlacionar bem com o comportamento médio de uma carga de um processamento de larga escala, os testes de avaliação pequenos não captam ou duplicam tais condições importantes de nível de sistema, como o tempo gasto em trocas de processos e interrupções de E/S. Para sistemas com caches, os programas de testes pequenos podem preenchê-los inteiramente, exagerando o aumento de desempenho de tais caches[32].

Como Levy e Clark declaram[22], vários outros efeitos sutis estão presentes quando os testes de avaliação de alto nível são usados para comparar sistemas. Por exemplo, eles argumentam que testes de avaliação implementados em linguagens diferentes não podem ser usados para se tirar conclusões de arquitetura. Como exemplo de como a semântica de linguagem pode afetar o resultado, eles observaram o sistema de manipulação de seqüência de caracteres em C (são usados ponteiros para acessar caracteres) *versus* Pascal (que indexa dentro de um vetor de caracteres), e dizem que, em linguagens como o Bliss PL/I, a instrução de combinação de séries de caracteres do VAX (Match C) poderia ser usada tendo-se um aumento de velocidade com um fator de aproximadamente 5.

Outros problemas com teste de avaliação de linguagem de alto nível relacionam-se com a qualidade do código do compilador (uma questão muito significante para o Intel 432, como será discutido na Seção 2.7). Clark e Levy mostraram exemplos de va-

riação de tempo de execução, em mais de 2:1, para compiladores diferentes da mesma linguagem em uma única arquitetura.

Outra questão na medição de desempenho de sistemas concerne à carga inserida pela execução do código do sistema operacional. Desde que exista uma variada gama de usos de funções do sistema operacional, pode ser muito difícil caracterizar tal sobrecarga, mas freqüentemente é estimado que um número substancial (mais que 50%) dos ciclos do processador são tipicamente dedicados ao sistema operacional. Sobrecargas de processamento dessa magnitude não podem ser ignoradas.

A despeito dos problemas associados com o uso de testes de avaliação, seu uso pode ser de grande valor em projetos de arquitetura. Em discussão sobre o desempenho do sistema LISP, Gabriel e masinter[13] defendem uma avaliação de arquitetura baseada em testes de avaliação de desempenho combinados com análise de mecanismos e estrutura.

"Arquiteturas de computadores tornaram-se complexas o suficiente para que geralmente seja difícil analisar o comportamento de programas sem um conjunto de testes de avaliação para guiar a análise. É difícil, geralmente, fazer uma análise acurada sem um trabalho experimental para guiar a análise e mantê-la acurada; sem análise é difícil saber como avaliar corretamente."

Temos a posição de que, embora haja vários problemas com a medição de sistemas usando-se testes de avaliação, ainda há boas razões para usá-los. Se os testes de avaliação forem obrigados a ser implementados em uma linguagem simples, e para uma arquitetura simples que é alterada de vários modos, então é possível obter-se conclusões não ambíguas sobre os efeitos daquelas decisões de arquitetura.

O trabalho RISC geralmente escolhe por trocar os benefícios tradicionais de uma dicotomia de arquitetura/implementação por uma flexibilidade de implementação que pode tornar uma avaliação de arquitetura mais difícil. Entretanto, prover alguns poucos exemplos de onde os projetistas do CISC podem ter ido longe na otimização de arquitetura e perdendo na eficiência de implementação não implica que os problemas sejam inevitáveis. Tem sido afirmado que avaliação de arquitetura é "estupidez", porque um estudo destes ignorou os efeitos da pré-aquisição de bytes não usados (um problema largamente evitado pela técnica de *desvio postergado* comum ao RISC I, MIPS e ao 801)[25]. Esse argumento é ardiloso e pode ser tratado de várias maneiras:

1. Os estudos poderiam ter incluído aqueles bytes.

2. Qualquer máquina, RISC ou CISC, pode usar desvios retartados (o 432 usa desvios retardados em suas rotinas no microcódigo); portanto este ponto é irrelevante.

3. Um estudo que erra, não invalida o método do qual ele deriva.

4. Uma máquina mais rápida não é necessariamente a que tem melhor arquitetura ou a que oferece utilidade ótima ou custo de ciclo de vida mais baixo, medidas que moram no coração dos estudos da arquitetura, mas são ignoradas nas publicações RISC.
5. Sem medidas de arquitetura não há meios para comparar as famílias de processadores de vários fabricantes; portanto não se pode abstrair os artefatos de implementação das inovações. O que realmente é necessário são medidas melhores que combinem ambas arquitetura/implementação para refletir melhor o custo de ciclo de vida esperado.

Muitos dos trabalhos RISC correntes são relevantes e úteis para projetistas de computadores, mas são melhor absorvidos em contextos derivados de qualquer outra parte. A tendência RISC não é a análise de problemas práticos enfrentados por fabricantes que precisam criar, produzir e dar suporte a uma linha de processadores com várias linguagens e sistemas operacionais. O trabalho RISC objetiva os aspectos de desempenho de baixo nível de implementação de computadores dentro de um restrito campo de utilização. Ele pode criticar os erros do CISC, mas oferece em troca somente um desempenho mais alto em operadores simples. Discussões que implicam mostrar que máquinas complexas, como o VAX, são mais rápidas quando somente suas instruções simples são usadas mostram apenas a inadequação dos compiladores atuais, ou os erros embutidos no microcódigo das instruções, tornando-as mais vagarosas do que as suas especificações de projeto determinam. Tais discussões não trazem luz aos problemas de desempenho associados ao projeto de uma máquina com microcódigo.

Os proponentes do RISC argumentam que microinstruções podem não ser mais rápidas que instruções simples, desde que isto implique que as instruções simples não sejam tão rápidas quanto poderiam ser. Acreditamos que esse argumento tem mérito, mas suas implicações não são o que parecem. Essa premissa tem sido usada para se concluir que a técnica de microcódigo é contraproducente desde que uma máquina de aplicação geral possa ser projetada para otimizar a execução de operações que contribuam mais para o desempenho global (as operações simples), e microcódigo não é necessário para isto. Diante de tal opção o objetivo do RISC é fortemente considerado.

Mas a premissa oculta é aquela de que tal opção deve ser feita. Não há razão para que uma realização CISC não possa descasar uma máquina tipo RISC necessária para operações simples de um microcódigo necessário para operações mais complexas (por exemplo, instrução *Send* do 432). De fato, este é precisamente o fato invocado pelos proponentes do RISC para ponto flutuante; porque não considerar isso para outras funções também? O ponto repetidamente discutido em publicações RISC é que "microcódigo não é mágico", isto é, tudo que o microcódigo pode fazer o software também pode. O mi-

crocódigo não é magica, mas nem é isomórfico ao software. De fato, há pelo menos dois caminhos importantes nos quais as funções implementadas em microcódigo diferem de seu equivalente em software.

A primeira diferença é a segurança. As máquinas de Von Newman colocam dados e instruções na mesma memória. Os mecanismos de arquitetura tentam regular o acesso a vários tipos de informação contidos na memória, mas essa proteção é necessariamente de um nível bastante tênue. Mesmo para uma máquina com domínio de proteção refinado como o 432 é possível intencionalmente ou acidentalmente usar erroneamente a memória. Quando uma função é alocada em microcódigo, a operação daquela função não pode ser alterada ou subvertida por alterações em suas instruções ou em dados imediatos. O 432 usa seu microcódigo para implementar corretamente seus mecanismos de endereçamento e proteção, que servem de base para todas as computações da máquina. Com um núcleo seguro, onde nenhuma atividade pode alterá-lo, é possível criar um sistema operacional confiável. Tal perspectiva não é praticável para máquinas onde um conjunto de bits na memória guarda outro conjunto de bits.

Outra diferença entre microcódigo e software pode parecer mágica se for dado o enfoque certo. Dada uma máquina que pode ser construída de forma que seu microcódigo não diminua a velocidade de execução de seus operadores simples, qualquer que seja a funcionalidade que resida no microcódigo, esta pode ser vista como uma rotina de software que tenha sido instantaneamente carregado no cache de instruções e que ao mesmo tempo tornou-se grande o suficiente para suportar a rotina. Essa rotina foi então executada sem provocar uma falta sequer no cache, resultando em nenhum acesso à memória, tendo-se terminado o cache exatamente como estava no início da rotina. Naturalmente a memória de microcódigo é bem mais eficiente que um cache, pois memória apenas de leitura é mais densa, podendo armazenar funções maiores que se o programa realmente tivesse de ser executado em um cache. As questões em aberto aqui são aquelas para determinar as funções que seriam apropriadas para implementação de co-processadores e para encontrar os mecanismos mais eficientes de comunicação entre o processador e seus co-processadores. Deve-se lembrar que um co-processador pode ou não residir na mesma pastilha que o processador central; onde a tecnologia permitir, este pode ser um uso mais adequado da pastilha do que caches grandes ou conjuntos de registradores largos.

2.6 O ESTUDO DO CONJUNTO DE MÚLTIPLOS REGISTRADORES

Provavelmente o processador do estilo RISC mais publicado é o RISC I da Berkeley. A característica mas bem conhecida dessa pastilha é seu grande número de registradores em pastilha organizados em uma série de conjuntos de registradores sobrepostos.

Entretanto, isto é irônico, pois conjuntos de registadores é uma característica para desempenho independente de qualquer aspecto RISC (como definido anteriormente) do processador. Os conjuntos de múltiplos registradores (MRS) podem ser incluídos em qualquer máquina de uso geral baseada em registradores.

Tem-se afirmado que a pequena área necessária para o hardware de controle para implementar um conjunto de instruções simples de um RISC em VLSI deixa uma boa área livre para implementar um conjunto de registradores grande[24]. A quantidade relativamente pequena de lógica de controle usada pelo RISC realmente libera recursos (em qualquer tecnologia) para outros usos, mas um conjunto de registradores grande não é o único caminho para usá-los, ou mesmo necessariamente o melhor. Por exemplo, o 801 e o MIPS escolheram outros caminhos para usar o hardware disponível; esses RISC tinham somente um conjunto de registradores simples e de tamanho convencional. Dos vários usos possíveis para tais recursos "liberados" por um conjunto de instruções RISC simples há alguns como caches, hardware para ponto flutuante e suporte para comunicação interprocessos, entre outros. Mais além, à medida que a tecnologia avança, os compromissos entre a complexidade do conjunto de instruções e as características de arquitetura/implementação tornam-se mais restritos. Os projetistas de computadores terão sempre de decidir qual o melhor uso de seus recursos disponíveis, mas dentro disso eles deverão perceber quais inter-relações são intrínsecas e quais não são.

Os estudos da Berkeley, descrevendo os processadores RISC I e RISC II, afirmam que suas decisões em relação aos recursos produziram um grande benefício de desempenho (2 a 4 vezes) em relação às máquinas CISC, como o VAX e o 68000[24,26]. Esses estudos, entretanto, não separam os efeitos de desempenho do conjunto de instruções reduzido dos efeitos dos conjuntos de registradores sobrepostos. Sendo assim, deduz-se que o desempenho derivado de características RISC dessas máquinas não foi demonstrado naqueles estudos, porque as características diferentes de desempenho não eram avaliadas independentemente.

Algumas comparações de desempenho entre máquinas diferentes, especialmente as mais antigas, foram baseadas em tempos de execução dos testes de avaliação. Enquanto a velocidade absoluta é sempre interessante, outras medidas menos dependentes de implementação podem prover informações de projeto mais úteis para arquitetos de computador, tal como o tráfego processador-memória, necessário para executar uma série de testes de avaliação. É difícil também determinar respostas seguras, a não ser que algum trabalho tenha sido feito para fatorar características dependentes de implementação não comparadas (por exemplo, caches e aceleradores de ponto flutuante).

Baseando-se nessas reservas, conduziram-se experimentos na CMU para testar a

hipótese de que os efeitos dos conjuntos de múltiplos registradores (MRS) são ortogonais em relação à complexidade do conjunto de instruções[10]. O objetivo era averiguar se os efeitos de desempenho dos MRS eram comparáveis para RISC e CISC. Para esses experimentos simuladores foram escritos para dois CISC (VAX e 68000), sem MRS, com MRS não sobrepostos e com MRS com sobreposição. Também foram escritos simuladores para o RISC I; RISC I com MRS de registros não sobrepostos, e RISC I com conjunto simples de registradores. Em cada um dos simuladores foi tomado o cuidado para não alterar a arquitetura inicial mais do que o absolutamente necessário para acrescentar ou remover os MRS. Em vez de simular tempo de execução, o tráfego processador-memória (bytes lidos e escritos) para cada teste de avaliação foi gravado como medida de comparação. Para usar esses dados razoavelmente, foram comparadas somente versões diferentes de conjuntos de registradores da mesma arquitetura (impedindo algumas ambigüidades que surgem em comparações de máquinas vastamente diferentes como o RISC I e o VAX). Os testes de avaliação usados foram os mesmos que originalmente avaliaram o RISC I. Um sumário dos experimentos e seus resultados podem ser encontrados em um artigo recente[20].

Como esperado, os resultados mostram uma diferença substancial no tráfego processador-memória para uma arquitetura com e sem MRS. As versões de MRS do VAX e o do 68000 mostram um decréscimo no tráfego para testes de avaliação altamente procedurais, como visto nas Figuras 2.4 e 2.5. Similarmente, as versões de conjunto de registradores simples do RISC I requerem muito mais memória lida e escrita do que o RISC I com conjuntos de registadores sobrepostos (Figura 2.6). Isto é devido em parte ao método de manipulação de estouro do conjunto de registros, que foi mantido para as três variações. Usando-se um esquema mais inteligente, o conjunto de registradores simples do RISC I realmente requer menos bytes no tráfego de memória na função de Ackermann que seu equivalente com conjunto de registradores múltiplos (veja a Figura 2.7). Para testes de avaliação com poucas chamadas de procedimentos (por exemplo, o Crivo de Eratostenes), o conjunto de registradores simples tem a mesma quantidade de tráfego processador-memória que a versão de MRS da mesma arquitetura. Novamente, um artigo recente descreve esse resultado, e outros[20].

Mais claramente, os MRS podem afetar a quantidade de tráfego processador-memória necessária para executar um programa. Em grande parte, o desempenho do RISC I em ambientes de procedimentos intensivos pode ser atribuído a seu esquema de conjuntos de registradores sobrepostos, uma característica independente da complexidade do conjunto de instruções. Dessa forma, qualquer desempenho reivindicado por computadores com conjunto de instruções reduzido que não remova efeitos decorrentes dos conjuntos de registradores múltiplos é inconclusivo.

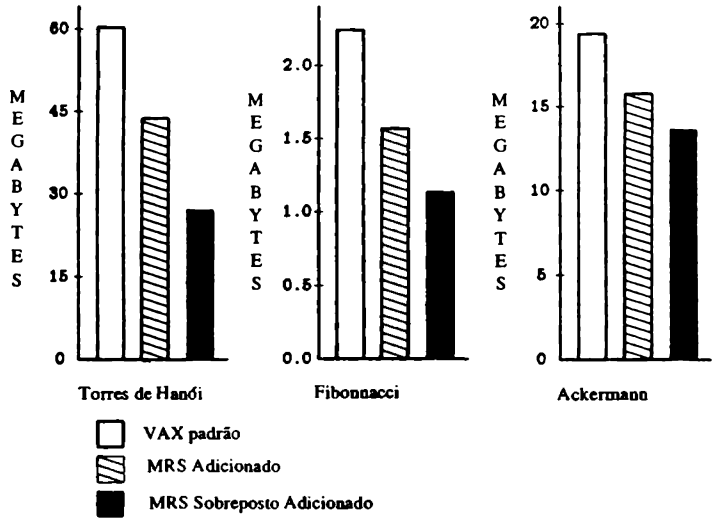


Figura 2.4 Tráfego memória-processador total para testes de desempenho em um VAX padrão e dois VAX modificados, um com conjunto de múltiplos registradores e outro com conjunto de múltiplos registradores sobrepostos.

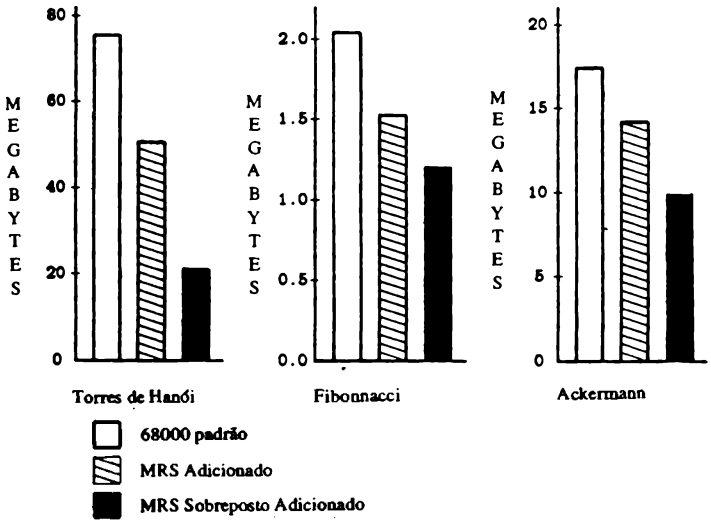


Figura 2.5 Tráfego memória-processador total para testes de desempenho para 68000 padrão e dois 68000 modificados, um com conjunto de múltiplos registradores e outro com conjunto de múltiplos registradores sobrepostos.

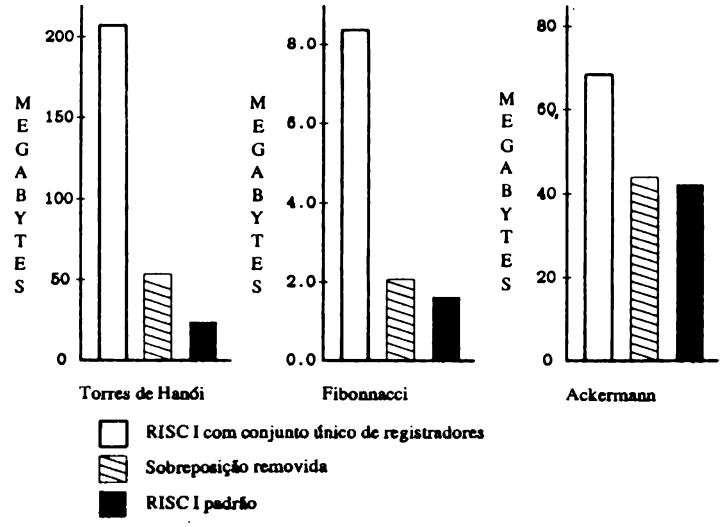


Figura 2.6 Tráfego memória-processador total para testes de desempenho em um RISC I padrão e em dois RISC I modificados, um com nenhuma sobreposição entre conjunto de registradores e outro com apenas um conjunto de registradores.

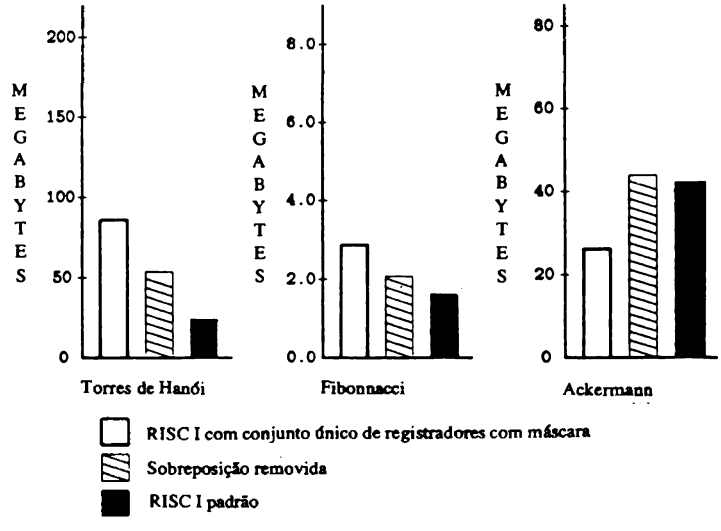


Figura 2.7 Tráfego memória-processador total para testes de desempenho com um RISC I padrão e dois RISC I modificados, um sem sobreposição entre conjuntos de registradores e outro com um conjunto único de registradores que utiliza uma máscara de registradores para indicar quais registradores devem ser salvos e restaurados em chamadas e retornos de procedimentos.

Esses experimentos da CMU usaram testes de avaliação programados para outros estudos do RISC com a finalidade de continuidade e consistência. Alguns desses testes, tal como o Ackermann, Fibonnacci e Hanói, realmente usaram mais tempo nas chamadas de procedimentos. A percentagem de tráfego processador-memória total devido a chamadas de procedimento do C para esses testes com conjuntos de registradores simples no 68000 é de 66 a 92%. Não foi uma surpresa que uma máquina como o RISC I, com uma estrutura de registradores sobrepostos, aos quais permitem as chamadas a procedimentos serem quase livres em termos de tráfego no barramento processador-memória, faria extremamente bem esses testes altamente recursivos quando comparados a máquinas com apenas um conjunto de registradores simples. Entretanto, não está estabelecido que esses testes são representativos de qualquer ambiente de computação.

2.7 O INTEL 432

Como um exemplo clássico de um CISC, o Intel 432 tem poucos semelhantes (compatíveis). Ele é um conjunto de circuitos integrados de microprocessadores VLSI orientado a objeto, projetado expressamente para fornecer um meio de programação Ada de grande escala, multiprocessamento e sistemas de multiprocessadores. A natureza de seu suporte arquitetural para objetos é tal que todo objeto é protegido uniformemente sem considerar distinções tradicionais como modo supervisor/usuário ou estruturas de dados de sistema/usuário. O 432 tem um conjunto de instruções muito complexo. Suas instruções são bit-codificadas também, e seu tamanho está entre o intervalo de 6 a 321 bits. O 432 incorpora um grau significante de migração funcional do software para microcódigo (a primitiva SEND de comunicação interprocessos é um exemplo de instrução do 432).

Os estudos publicados sobre o desempenho do 432 da Intel em testes de avaliação do baixo nível (por exemplo, Torre de Hanói[16]) mostram que ele é bem lento, precisando aproximadamente de um tempo de 10 a 20 vezes superior ao VAX 11/780. Tal projeto poderia parecer um candidato ideal para investigação da controvérsia RISC/CISC.

Intuitivamente, pode-se tentar criticar o tempo de execução no ambiente orientado a objetos das máquinas, por causa de seu alto custo. Toda referência à memória é verificada para garantir que ela se mantenha dentro dos limites do objeto referenciado, e as permissões de escrita/leitura do contexto em execução são verificadas para aquele objeto. Os proponentes do RISC argumentam que a complexidade da arquitetura do 432, mais a decodificação adicional necessária para o fluxo de instrução bit-codificado, contribuem para seu baixo desempenho. Para endereçar esses e outros eventos, foi empreendido um estudo detalhado do 432, o qual avaliou a eficiência dos mecanismos de arquitetura (migrações funcionais) que o 432 proveu no suporte de seu ambiente em tempo de execução

esperado. Esta é uma das diferenças centrais dos estilos de projeto RISC e CISC: projetos RISC evitam estruturas de hardware/microcódigo planejados para dar suporte ao ambiente de tempo de execução, procurando colocar tal suporte no compilador ou programa. Isto vai de encontro à tendência tradicional do projeto de conjunto de instruções de um computador, que reflete uma firme migração de funções de nível elevado (programacionais) para um nível baixo (hardware ou microcódigo), na expectativa de aumentar o desempenho.

Ao contrário da maior parte do trabalho RISC, esse estudo não assumiu que qualquer aspecto da arquitetura do sistema ou da implementação são candidatos para alterações ou remoção enquanto o desempenho total do sistema nas avaliações pareça ser elevado. O estudo persegue a questão de quão grande parece ser o custo inerente em um ambiente orientado a objeto, e como efetivamente uma migração funcional pode ser utilizada para combater tal custo. Conseqüentemente, foi analisado o fator de desempenho mais elevado em relação a aspectos de tempo de execução, que é intrínseco a sistemas orientados a objetos da classe do 432.

O desempenho do 432 foi avaliado usando um conjunto de testes que estimularam o microssimulador 432. Esse simulador gerou arquivos de acompanhamento ciclo por por ciclo, os quais foram analisados. Foram então propostas mudanças na arquitetura, usando-se como referência o número de ciclos por instrução fornecido pelos arquivos de acompanhamento. A manipulação detalhada desses experimentos e seus resultados podem ser encontrados em uma tese recente[11].

As simulações desses testes de avaliação revelaram vários problemas de desempenho com o 432 e seu compilador:

1. O compilador ADA do 432 não realiza quase nenhuma otimização. Como conseqüência, a máquina é forçada freqüentemente a fazer mudanças desnecessárias no seu complexo contexto de endereçamento, a alto custo são recomputadas subexpressões desnecessariamente, as passagens de parâmetros são feitas sempre por valor, que em muitos casos requer substancialmente mais tráfego na memória do que chamada por referência. Isto distorce seriamente muitos resultados das comparações de testes de avaliação. Tais testes realmente refletem o desempenho que se pode esperar da versão do 432 atual, mas dizem muito pouco sobre a eficácia dos compromissos de arquitetura dessa máquina.

2. A memória do 432 tem banda de memória bem limitada. Isto é resultado de vários fatores, especificamente o seguinte: o 432 não tem cache de dados na pastilha, nem fluxo de instruções literal, nem registradores de dados locais, o que o obriga a ter mais referências na memória do que teria de outra forma. Isto também torna o tamanho do código

bem maior, desde que não necessários muito mais bits para referenciar dados dentro de um objeto do que com um registrador local. Em virtude de suas limitações de pinagem, o 432 tem de multiplexar endereços e dados através de somente 16 pinos. Também o sistema de desenvolvimento Intel 432/600 padrão, que suporta multiprocessamento com memória compartilhada, usa um barramento assíncrono lento, que foi projetado mais pela integrabilidade do que pela eficiência. Todos esses aspectos de implementação combinados fazem com que o tempo do processador seja consumido em 25 a 40% do tempo de teste, por estados de espera.

3. Em testes de avaliação altamente recursivos, o custo do ambiente orientado a objetos no 432 aparece de fato na forma de chamadas de procedimento lentas. Mas mesmo aqui os problemas de desempenho não deveriam ser atribuídos ao ambiente orientado a objetos ou à complexidade intrínseca da máquina. Os projetistas do 432 tomaram uma decisão para fornecer um novo contexto protegido para toda chamada; o usuário não tem opção a esse respeito. Se um mecanismo de chamada não-protegido fosse usado apropriadamente, o teste de avaliação Dhrystone[32] rodaria vinte vezes mais rápido.

4. O alinhamento em bit das instruções significa que o 432 deve necessariamente decodificar os vários campos de uma instrução seqüencialmente. Desde que tal decodificação geralmente coincide com a execução da instrução, o 432 atrasa esperando pela decodificação em 3% do tempo nesses testes de avaliação. Entretanto, essa percentagem só cairá quando os problemas acima forem eliminados.

Várias degradações de ciclo mencionadas acima têm impacto significante no desempenho do 432 como um todo, elas ainda não estão relacionadas à complexidade da arquitetura, migração funcional, ou orientado a objeto. Conseqüentemente, várias otimizações de arquitetura foram exploradas para determinar o número de ciclos realmente gastos durante a execução. Essas otimizações são:

- melhor gerenciamento dos ambientes de endereçamento (interambientes);
- melhor otimização do código pelo compilador;
- mecanismos apropriados usados para chamadas de procedimentos (chamadas protegidas *versus* desvios com enlace);
- uso de mecanismos mais rápidos disponíveis para passagem de parâmetros;
- codificação de instruções não-orientadas em bits;
- caches para literais no conjunto de instruções.

A Tabela 2.2 mostra os ciclos salvos quando as suposições forem feitas.

Tabela 2.2 Novos ciclos-base e aumento percentual de desempenho em relação ao original

Testes de desempenho	*Ciclos ganhos*	*Ciclos originais salvos*	*Ciclos-base sintéticos*
		%	
Acker	8 864 736	2,2	385 785 657
Crivo	1 130 839	14,9	6 472 647
CFA5	15 228 248	43,6	19 688 197
CFA5R	24 207 058	39,6	36 930 835
CFA10	21 795 608	44,7	27 007 880
Dhrystone	655 452	93,7	44 168

As induções da Tabela 2.2 devem ser claramente compreendidas. Esta tabela mostra que foram desperdiçados de 35 a 40% do total de ciclos de execução dos testes de avaliação. Esses ciclos não são usados em atividades no ambiente orientado a objetos; não são uma conseqüência inevitável de um conjunto de instruções complexo; eles não refletem a ineficiência alegada de um processador microdecodificado. Afirmamos que esses ciclos são consumidos por causa de decisões de projeto subótimas ou erros, os quais poderiam ter sido cometidos em qualquer novo projeto de sistema, baseado em objeto ou não.

As contribuições relativas de cada uma das otimizações discriminadas acima são mostradas como percentagens na Tabela 2.3. Note que essas percentagens representam os salvamentos de ciclos quando cada otimização é considerada individualmente, enquanto a Tabela 2.2 representa os efeitos combinados de todas as otimizações de arquitetura (esta é a razão pela qual as percentagens da Tabela 2.2 não refletem a Tabela 2.3).

Tabela 2.3 Contribuições relativas de melhoria sobre a base original, em percentagens

Testes de desempenho	*Entrada*	*Código operação*	*Chamada proc.*	*Parâmetros*	*Alinhamento*	*Constantes*
	%	%	%	%	%	%
Acker	0	0	0	0	67	33
Crivo	0	0	0	0	60	40
CFA5	44	27	12	6	5	6
CFA5R	25	19	8	39	5	4
CFA10	35	17	8	26	8	6
Dhrystone	1,2	0	1,8	90,0	0,2	0,3

Desde que toda categoria na Tabela 2.3 contribui substancialmente para aumen-

tar a velocidade de um dos testes de avaliação, esse dado sugere que cada um desses melhoramentos é significativo e poderia ter sido incorporado no 432 originalmente. Os tempos de execução do Acker e do Crivo não são vastamente otimizados pelo alinhamento e a literação das instruções, mas os ciclos nunca deveriam ser gastos em tais elementos fundamentais de uma arquitetura, a não ser em um caso muito especial que não o desempenho; aqui ele não pode.

Esses fatores que contam para o baixo desempenho têm de ser estabelecidos, analisados e removidos das considerações de arquitetura, pois são irrelevantes para a análise de arquitetura de migração funcional. O próximo passo dado para determinar os custos de desempenho necessariamente sujeitos ao suporte ao tempo de execução do 432, em ambiente orientado a objeto, foi ordenar os ganhos de desempenho possíveis se uma tecnologia melhor (tamanho característico menor, por exemplo, estivesse disponível para uma inovação do 432. Tal informação sugeriria o preço total para orientação a objeto para uma tecnologia implementada contemporaneamente.

Os seguintes benefícios foram considerados para o 432:

- existência de registradores de dados locais;
- expansão de barramentos internos e externos de 16 para 32 bits;
- expansão do registrador da pilha de 16 bits para 32 bits;
- um bit extra para a microinstrução de barramento;
- um cache para descritor de memória (AD-cache);
- uma operação primitiva para limpar a memória.

Tabela 2.4 Ciclos salvos com a melhoria da tecnologia de implementação por percentagens

Testes de desempenho	*Reg. de dados*	*Barramento 32 bits*	*Reg. de pilha de 32 bits*	*μ instrução de 17 bits*	*Cache descritos de endereço*	*Limpeza de memória*
	%	%	%	%	%	%
Acker	0	45	0	1	12	42
Crivo	82	18	0	0	0	0
CFA5	34	32	17	0	15	2
CFA5R	19	47	15	1	17	1
CFA10	40	35	3	0	20	2
Dhrystone	8	43	1	1	26	22

A Tabela 2.5 mostra o total de benefícios nos ciclos em percentagens, se esses melhoramentos forem incorporados. A Tabela 2.4 mostra como cada benefício muda para cada teste de avaliação. Note novamente que às alterações entre os benefícios contam com diferenças de somas das percentagens nessas tabelas.

Tabela 2.5 Melhoria sobre o original em ciclos e percentagens

Testes de desempenho	*Ciclos ganhos*	*Base original de ciclos ganhos*	*Aumento tecnológico de ciclos*
		%	
Acker	130 452 160	33	264 021 113
Crivo	4 489 605	59	4 489 605
CFA5	15 212 797	44	19 692 875
CFA5R	25 998 902	43	25 983 605
CFA10	21 027 060	43	27 769 185
Dhrystone	23 866	35	45 150

A Tabela 2.6 mostra os efeitos combinados de todas as alterações na arquitetura, tecnologias de compilador e implementação listadas acima.

Tabela 2.6 Total sintético de ciclos básicos, e percentagens ganhas sobre a base real, e tempo real em milissegundos

Teste de desempenho	*Total final de ciclos*	*Base original de ciclos ganhos*	*Tempo real*
		%	MS
Acker	257 319 033	35	32 165
Crivo	2 653 785	65	332
CFA5	11 025 390	68	1 378
CFA5R	21 050 576	66	2 631
CFA10	15 394 492	68	1 924
Dhrystone	28 709	58	3,59

Os projetistas do 432 têm afirmado que, para testes de avaliação de limites de computação (como o Crivo), o 432 poderia não exibir obrigações de desempenho maiores, uma vez que as operações de orientação a objeto tenham sido feitas. A melhoria de 65% apresentada para o program Crivo eleva seu desempenho a um nível onde é ele compatível com outras máquinas, como o 68000, da Motorola, e 8086, da Intel, e como tal provê a primeira evidência direta para tal afirmativa.

O Acker representa o pior caso para qualquer máquina orientada a objeto, executando operações apenas nas quais ele é vagaroso (chamadas de procedimento). Como tal, é considerado que para códigos com intensivas chamadas a procedimentos o custo do ambiente orientado a objeto tem um fator de quatro.

Esses resultados estabelecem uma posição na qual o custo do ambiente orientado a objeto demonstrado pelo 432 pode ser avaliado. Além disso, o preço para objeto orientado parece variar no fator de 1 para 4 (Crivo até Acker) acima dos preços de arquiteturas convencionais.

O experimento do 432 evidencia, no mínimo para teste de avaliação de computação de baixo nível, a ênfase renovada do RISC sobre a importância de decodificação rápida de instruções (formatos de instruções regulares e fixos) e armazenamento local rápido (tal como caches e registradores). Tem sido demonstrado que o 432 paga taxas significativas de desempenho em virtude da falta de cache de dados em pastilha, e instruções orientadas a bit de tamanho variável. O teste Crivo adquiriu velocidades competitivas quando justamente seus oito registradores foram incorporados ao 432. O teste de avaliação aumentou sua velocidade em 8% quando os ciclos gastos usados na decodificação de instruções foram removidos. É mais importante notar aqui que esses erros são conseqüência inevitável do projeto de computador com conjunto complexo de instruções. Não há razão que indique que as mesmas funcionalidades do conjunto de instruções do 432 não sejam realizadas em uma máquina com uma codificação de instrução mais regular e que possua memória local rápida.

Quando se comenta sobre o desempenho de um sistema de computador, é muito importante atribuir ao desempenho divulgado (mau ou bom) a sua causa. O 432 exibe um desempenho pobre devido ao ambiente de execução adotado e a visíveis erros de projeto, não porque seja um computador com conjunto complexo de instruções.

As preocupações do RISC com "chamadas" e "retornos" de procedimentos rápidos são justificadas no 432, mas não são necessariamente soluções RISC. O 432 sofre comparativamente com "chamadas" e "retornos" de procedimentos lentos, que podem torná-lo aproximadamente quatro vezes mais lento que um processador convencional. Entretanto, a proteção oferecida pelo paradigma baseado em objeto pode ser tal que o preço sobre o desempenho seja aceitável. Quando os tempos de execução de máquinas com orientação a objeto são comparados aos de máquinas mais convencionais, deve-se lembrar que estão sendo feitas uma gama diferente de trabalho nas respectivas máquinas. Sistemas orientados a objeto trocam desempenho em função de um meio de programação potencial mais produtivo, o que é importante se considerar quando se faz comparações entre tais sistemas de computadores diferentes.

Embora o 432 exiba um desempenho pobre, não se pode considerar evidente que a migração de funções para microcódigo e hardware em larga escala seja ineficiente. Ao contrário, Cox e outros[12] demonstram que a implementação de microcódigo no 432 para comunicação interprocesso, por exemplo, é mais rápida que a versão de software equivalente. Em testes de avaliação de baixo nível usados nesses estudos, o 432 poderia ter chegado a um desempenho muito mais alto somente com pequenas mudanças em sua implementação e um compilador melhor. Com atenção a todos esses pontos seria errado concluir que o 432 constitui uma demonstração para a ótica RISC.

2.8 SUMÁRIO

Neste capítulo, várias perspectivas do RISC e CISC foram exploradas. Examinando-se declarações, história, filosofia, desafios, comercialização, confusões e reavaliações do RISC, foi dada uma perspectiva ampla em projetos de computadores. É uma perspectiva que não dicotomiza o RISC e o CISC, criando divisões não-naturais e respostas monolíticas. Ao contrário, ele busca entender o espaço de projeto de computadores examinando características individuais que compõem uma máquina. É uma perspectiva preocupada com o mapeamento de funções para um nível de implementação próprio, indiferente ao rótulo designado ao projeto resultante.

RISC, sem dúvida, se tornará uma parte importante do cenário de computadores num futuro próximo. Ainda não está claro, contudo, se tais projetos eventualmente dominarão a arquitetura de computadores ou não. Desconsiderando-se isso, alguns benefícios da evolução RISC já são aparentes. Os projetos RISC têm se confrontado com os estilos de projetos tradicionais. Dentro disso, a filosofia RISC questiona várias suposições básicas e faz algumas afirmações perspicazes e atrevidas. Isto tem trazido auto-examinação em vários aspectos de projeto de computador; um resultado que é sempre bem-vindo.

REFERÊNCIAS BIBLIOGRÁFICAS

1. "The IBM System/360 Model 91: Machine Philosophy and Instruction Handling", Anderson, D. W., Sparacio, F. A. e Tomasulo, R. M. *IBM J. Res. & Dev.*, **11**, 1, 8-24, jan. 1967.
2. *The Acorn RISC Machine*. Acorn Computers Ltd., Fulborn Road, Cherry Hinton, Cambridge, England, CB1 4JN, 1985.
3. "The MODHEL Microcomputer for RISCS Study", Azaria, Helnye and Tabak,Daniel. *Microprocessing and Microprogramming*, **12**, 3–4; 199-206, out./nov. 1983.
4. "Sentry: A Novel Hardware Implementation of Classic Operating System Mechanisms", Bar-

ton, G. C. *Proc. 9th Ann. Symp. Comp. Arch.*, IEEE/ACM, Austin, Texas, 140-147, abr. 1982.

5. "The Operating System and Language Support Features of the BELLMAC-32 Microprocessor", Berenbaum, Allen D., Condry, Michael W. e Lu, Priscilla M. *Proc. Symp. Arch. Support for Prog. Lang. and Op. Syst.*, ACM, Palo Alto, California, 30-38, 1-3, mar. 1982.
6. "Understanding Execution Behavior of Software Systems", Browne, James C. *Computer* **17**, 7, 83-87, jul. 1984.
7. "Comments on 'The Case for the Reduced Instruction Set Computer', by Patterson and Ditzel", Clark, Douglas W. e Strecker, William D. *Comp. Arch. News*, **6**, 6, 34-38, out. 1980.
8. "Measurement and Analysis of Instruction Use in the Vax-11/780", Clark, Douglas W. and Levy, Henry M. *Proc. 9th Ann. Symp. Comp. Arch.*, IEEE e ACM, 9-17, abr. 1982.
9. "A Perspective on the Processor Complexity Controversy", Colwell, Robert P., Hitchcock III, Charles Y. e Jensen, E. Douglas. *Proc. Int. Conf. Comp. Des.: VLSI in Computers*, IEEE, Port Chester, New York, 613-616, 31 out.-3 nov. 1983.
10. "Peering Through the RISC/CISC Fog: An Outline of Research", Colwell, Robert P., Hitchcock III, Charles Y. e Jensen, E. Douglas. *Comp. Arch. News*, **11**, 1, 44-50, mar. 1983.
11. *The Performance Effects of Functional Migration and Architectural Complexity in Object-Oriented Systems*, Colwell, Robert P. PhD thesis, Carnegie-Mellon University, mai. 1985.
12. "Interprocess Communication and Processor Dispatching on the Intel 432", Cox, George W., Corwin, William M., Lai, Konrad K. e Pollack, Fred J. *ACM Trans. Comp. Sys.*, **1**, 1, 45-66, fev. 1983.
13. "Performance of Lisp Systems", Gabriel, Richard P. e Masinter, Larry M. *ACM Symp. Lisp and Func. Prog.*, ago. 1982.
14. "Windows of Overlapping Register Frames", Halbert, Daniel C. e Kessler, Peter B. *CS292R Final Reports*, University of California, Berkeley, 9 jun. 1980.
15. *Tutorial: The Migration of Function into Silicon*, Hammerstrom, Dan, given at the *10th Ann. Int. Symp. Comp. Arch.*
16. "A Performance Evaluation of the Intel iAPX 432", Hansen, Paul M., Linton, Mark A., Mayo, Robert N., Murphy, Marguerite and Patterson, David A. *Comp. Arch. News*, **10**, 4, 17-27, jun. 1982.
17. "MIPS: A VLSI Processor Architecture", Hennessy, John, Jouppi, Norman, Baskett, Forest and Gill, John. *Proc. CMU Conf. VLSI Sys. and Comps.*, 337-346, out. 1981.
18. "Hardware'Software Tradeoffs for Increased Performance", Hennessy, John, Jouppi, Norman, Baskett, Forest, Gross, Thomas and Gill, John. *proc. Symp. Arch. Supp. for Prog. Lang. and Op. Sys.*, ACM, Palo Alto, California, 2-11, 1-3 mar. 1982.
19. "VLSI Processor Architecture", Hennessy, John. *IEEE Trans. Comp.*, **C-33**, 12, 1221-1246, dez. 1984.
20. "Analyzing Multiple Register Sets", Hitchcock III, Charles Y. e Brinkley Sprunt, H. M. *Proc. 12th Int. Symp. Comp. Arch.*, IEEE/ACM, Boston, MA, 55-63, 17-19, jun. 1985.
21. "Money Starting To Flow As Parallel Processing Gets Hot", Iverson, Wesley R. *Electronics Week*, 36-38, abr. 1985.

22. "On the Use of Benchmarks for Measuring System Performance", Levy, Henry M. e Clark, Douglas W. *Comp. Arch. News*, **10**, 6, 5-8, dez. 1982.

23. "Messages and Multiprocessing in the ELXSI System 6400", Olson, Robert A., Kumar, B. e Shar, Leonard E. *Proc. Spring 1983 CompCon*, IEEE, mar. 1983.

24. "A VLSI RISC", Patterson, David A. e Sequin, Carlo H. *Computer*, **15**, 9, 8-21, set. 1982.

25. "Reduced Instruction Set Computers", Patterson, David A. *CACM*, **28**, 1, 8-21, jan. 1985.

26. "RISC Watch", Patterson, David. *Comp. Arch. News*, **12**, 1, 11-19, mar. 1984.

27. "The 801 Minicomputer", Radin, George. *Proc. Symp. Arch. Support for Prog. Lang. and Op. Sys.*, ACM, Palo Alto, California, 39-47, 1-3, mar. 1982.

28. "VAX-11/780: A Virtual Address Extension to the DEC PDP-11 Family", Strecker, William D. In *Computer Engineering: A DEC View of Hardware Systems Design*, Digital Press, Bedford, MA, Ch. 17, 409-428, 1978. VAX-11/780: A Virtual Address Extension to the DEC PDP-11 Family; reprinted in SBN.

29. "The Transputer", Whitby-Strevens, Colin. *Proc. 12th Int. Symp. Comp. Arch.*, IEEE/ACM, Boston, MA, 292-300, 17-19 jun. 1985.

30. "Architecture of SOAR: Smalltalk on a RISC", Ungar, David, Blau, Ricki, Foley, Peter, Samples, Dain e Patterson, David, *11th Ann. Int. Symp. Comp. Arch. Proc.*, IEEE/ACM, Ann Arbor, Michigan, 188-197, 5-7, jun. 1984.

31. "Design and Implementation of the VAX 8600 Pipeline", DeRosa, John, Glackemeyer, Richard e Knight, Thomas. *Computer*, **18**, 5, 38-48, mai. 1985.

32. "Dhrystone: A Synthetic Systems Programming Benchmark", Weicker, Reinhold P. *Comm. of the ACM*, **27**, 10, 1013-1030, out. 1984.

33. "Compilers and Computer Architecture", Wulf, William A. *Computer*, **14**, 7, 41-47, jul. 1981.

CAPÍTULO 3

O SISTEMA MICROPROCESSADOR DE 32 BITS DA AT&T WE32100

Priscilla M. Lu
AT&T, USA

3.1 INTRODUÇÃO

O WE32100 é a segunda geração de microprocessadores da AT&T. É complementado por quatro pastilhas periféricas, consistindo em suporte a gerenciamento de memória (MMU-WE32101), aceleração de aritmética de ponto flutuante (MAU-WE32106), controlador de acesso direto à memória (DMAC-WE32104) e controlador de acesso à memória dinâmica (DRC-WE32103). Fotografias desses cinco dispositivos são mostradas nas Figuras 3.1 a 3.5.

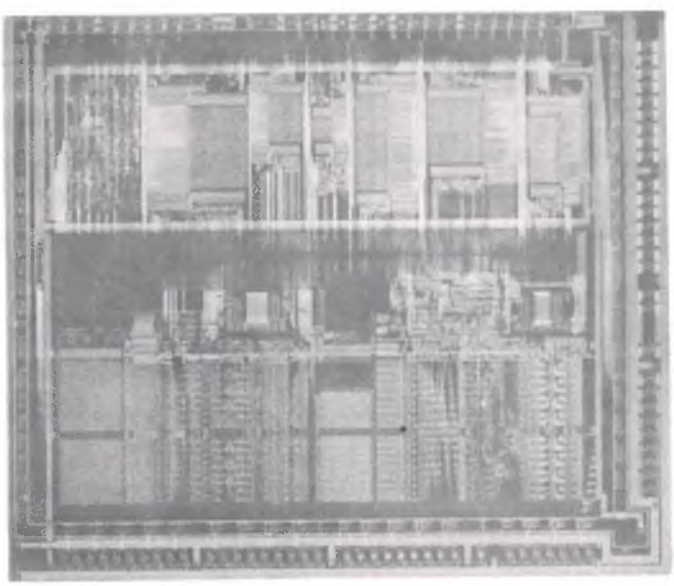


Figura 3.1 Microprocessador WE32100.

O conjunto de pastilhas forma os blocos básicos de construção de um sistema de 32 bits de alto desempenho. A arquitetura permite uma implantação eficiente do sistema operacional UNIX, suporte a linguagens de alto nível e a arquitetura de E/S foi concebida para simplificar o projeto de sistemas e maximizar o alto desempenho do sistema.

3.2 ARQUITETURA DO MICROPROCESSADOR WE32100

O WE32100 é implementado em tecnologia CMOS de 1,5 μm. A primeira geração WE32000[2] foi implementada em 1981, na tecnologia CMOS de 1,75 μm. O WE32100, verticalmente compatível com seu predecessor, possui como melhorias uma memória cache (I-cache) de 64 palavras e uma interface para co-processador de uso geral. A memória cache possui uma taxa de acerto de aproximadamente 65-70% para a maioria dos programas UNIX e contribui aproximadamente com 20-25% na melhora de desempenho de programas. O WE32100 opera a 18 MHz, com um desempenho geral de 2-3 MIPS.

O microprocessador possui um barramento bidirecional de 32 bits com decodificação de estado, arbitração de barramento para acesso externo, controle de DMA, processamento de interrupções, habilitador de "trace" e visibilidade dos pinos para facilitar testes e depurações.

O WE32100 dá suporte a quatro tipos de dados: bytes, meias palavras (half words), palavras (words) e campos de bits (de 1 a 32 bits). Bytes, halfwords e words podem ser interpretados como assinalados ou não assinalados em operações lógicas e aritméticas. Strings recebem suporte de um bloco especial de instruções (STRING COPY, STRING LENGTH). O formato dos strings é compatível com a linguagem C, terminada por um "null" ou byte zero.

Instruções são endereçáveis por byte e definidas por um código de 1 ou 2 bytes seguidos ou não por descritores de operandos. Todos os bytes ou halfwords quando operandos são estendidos com ou sem sinal durante a operação de leitura (fetch).

Os descritores de operando identificam a localização do operando. Existem diversos modos de endereçamento: literal, byte/halfword imediato, registrador, registrador referido, offset curto (para ponteiros de quadro ou argumento), deslocamento de byte/halfword/word, deslocamento referido de byte/halfword e operando expandido. Esses modos são cobertos com mais detalhes nas próximas seções.

Há um registrador contador de programa especial e quinze outros registradores no processador que podem ser referenciados em qualquer modo de endereçamento. Três

dos quinze registradores são privilegiados, isto é, só podem ser alterados quando o processador está no nível de execução núcleo. Eles são usados para dar suporte a operações dentro do sistema operacional. São usados como stack pointer (apontador de pilha), de interrupção, ponteiro a bloco de controle de processo e palavra de estado do processador. Outros três registradores são usados por instruções especiais como apontador de pilha, apontador de quadro e apontador de argumento. Um diagrama de blocos é mostrado na Figura 3.6

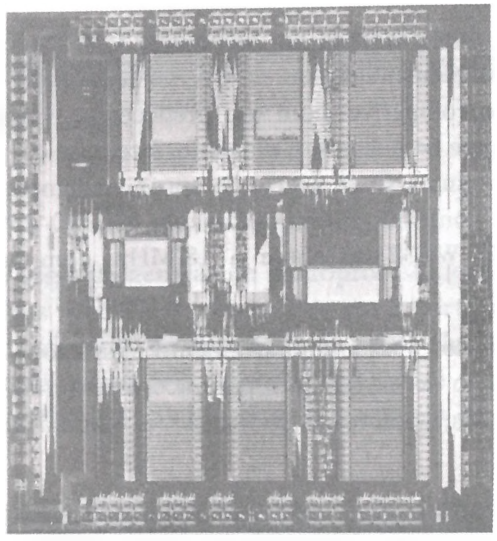


Figura 3.2 Pastilha de gerenciamento de memória WE32101.

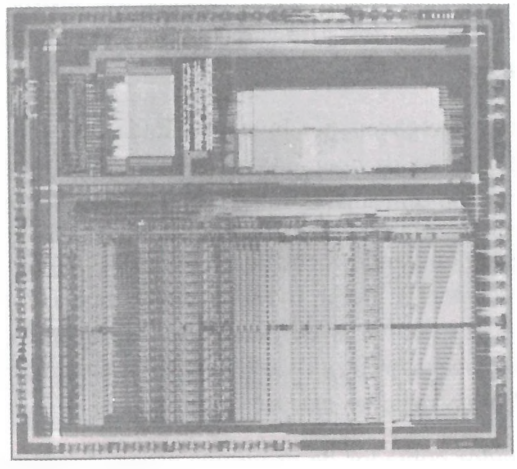


Figura 3.3 Pastilha aceleradora de aritmética em ponto flutuante WE32106.

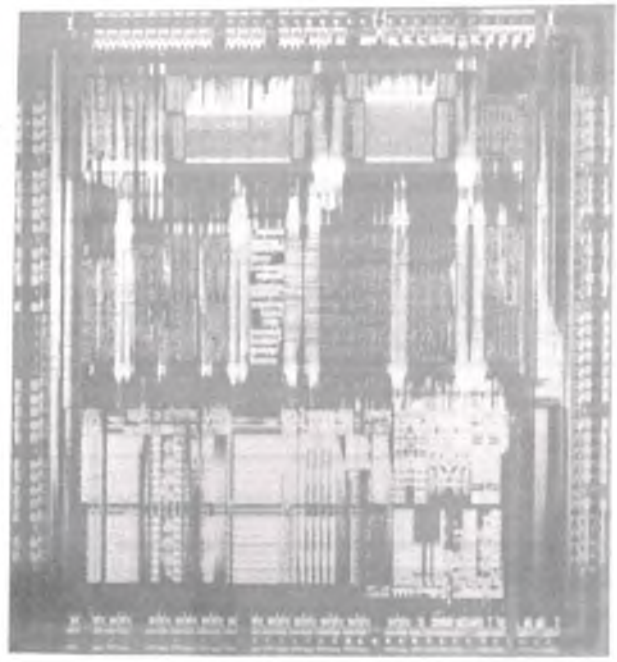


Figura 3.4 Contador de acesso direto à memória WE32104

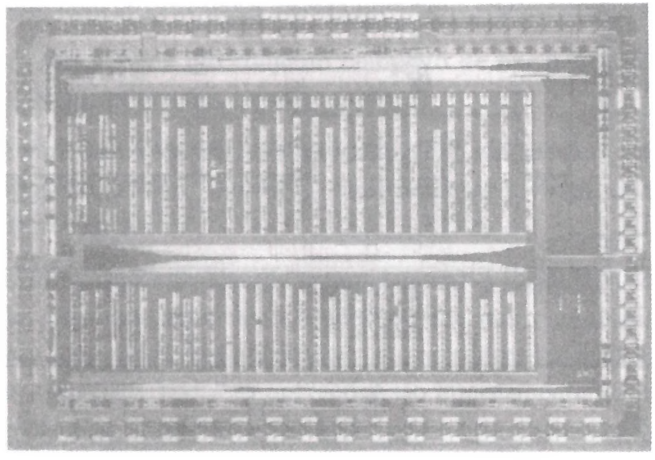


Figura 3.5 Controlador de memória de acesso direto dinâmica WE32103.

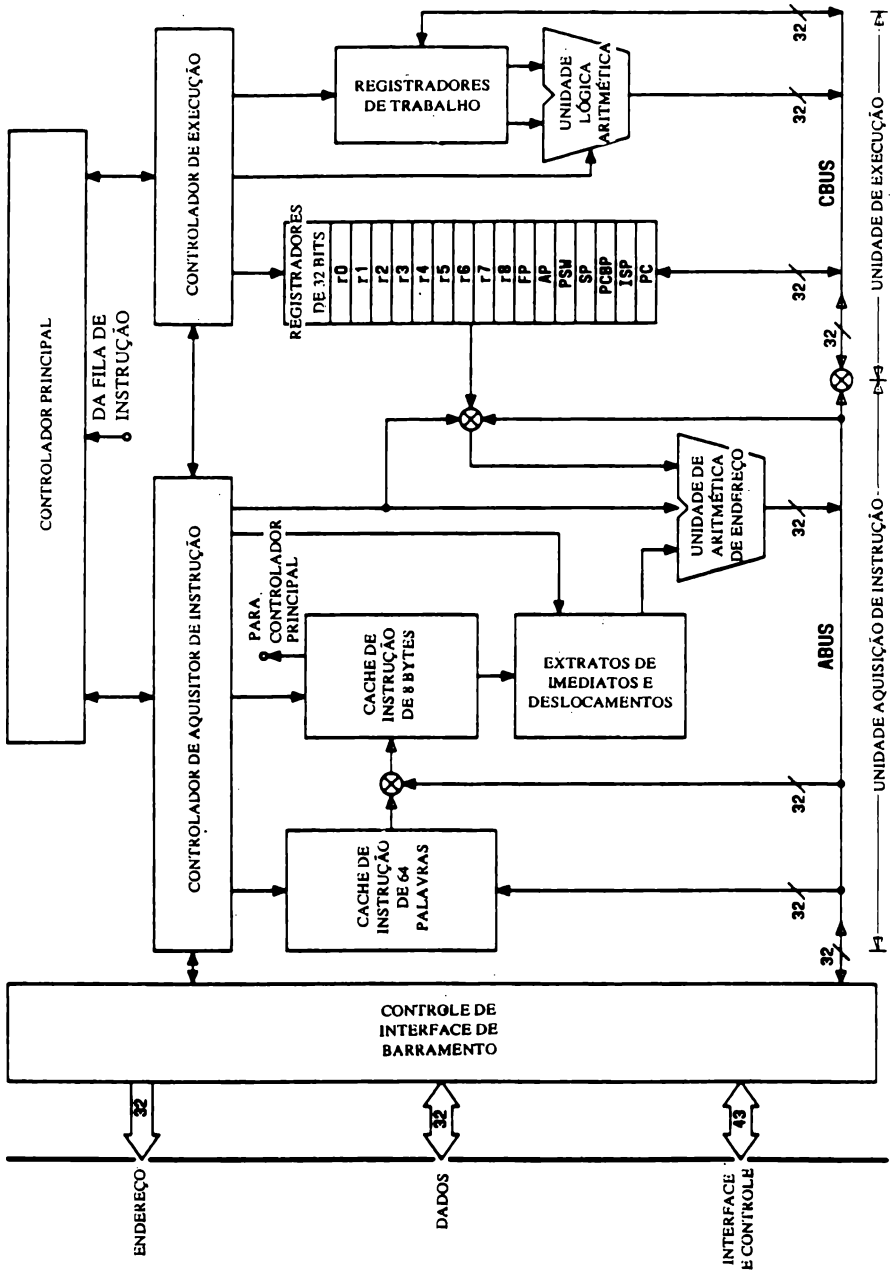


Figura 3.6 Diagrama de blocos do microprocessador WE32100.

3.3 SUPORTE À LINGUAGEM DE PROGRAMAÇÃO

Quando foi projetado o WE32100, um dos principais objetivos foi o suporte à linguagem C. A arquitetura resultante, no entanto, dá suporte às necessidades de linguagens de alto nível em geral, incluindo C. Características de suporte a linguagens incluem instruções úteis para implementação de operações lógicas e aritméticas, instruções especiais para manipulação de strings e campos de bits, operações tanto simples como de alto nível para chamadas de sub-rotinas. Muitas características do WE32100 simplificam a interface com o sistema operacional, incluindo o projeto da máquina "orientado a processo" como necessário para multiprogramação e um mecanismo especial de "transferência controlada" que implementa controle de exceções tanto definidas pelo usuário como pelo sistema.

3.3.1 INSTRUÇÕES LÓGICAS E ARITMÉTICAS

Instruções, endereçamento e até certo ponto os tipos de dados são ortogonais no WE32100. O código de operação define a função a ser executada, enquanto o descritor de operando (ou modo de endereçamento) especifica o tipo de dado. O descritor de operando pode ser um dos possíveis modos de endereçamento. Não há restrições de registrador ou tipo de dado nos operandos para qualquer operação. Instruções de máquina associam um tipo de dado default com os operandos se esse tipo não é especificado de outra forma.

O WE32100 oferece um conjunto completo de operações lógicas e aritméticas usuais. Resumidamente, as funções fornecidas são:

lógicas: complemento de um, OU inclusivo e OU exclusivo.
aritméticas: negação, adição, subtração, multiplicação, divisão, módulo, incremento e decremento.

O operador unitário de negação e complemento é formulado como uma instrução de movimentação; conseqüentemente o resultado pode tanto substituir o dado existente como ser colocado num novo destino. Todos os operadores binários possuem formas de instruções duplas ou triplas. Todas as operações são internamente executadas em 32 bits, porém ocorre *overflow* se o resultado obtido for de extensão maior que a do operando de saída.

O fato de todas as operações ocorrerem da mesma forma é conveniente para compiladores. Por exemplo, o uso de formas duplas ou triplas para calcular-se uma expressão é algumas vezes ignorado na otimização do compilador em virtude de restrições do operando.

A codificação da expressão em C

```
a = b + c * (d + e)
```

pode ser feita facilmente com três instruções (assuma todas as variáveis como words inteiras):

```
addw3    d,e,%r0     r0 = d + e
mulw2    c,%.r0      r0 = c * (d + e)
addw3    b,%r0,a     a = b + c * (d + e)
```

onde os nomes das variáveis representam algum descritor de operando para acessar a variável. A mesma seqüência poderia ser utilizada com diferentes operadores (binários) na expressão em C, com a correspondente substituição de *opcodes*, e as variáveis não precisam ser words inteiras.

Um modo de endereçamento de interesse particular é o *literal curto*, que pode representar um inteiro curto (entre –16 e 63) usando somente 1 byte para o dado e descritor. A utilização desse modo produz uma redução média de espaço de 5% e, como conseqüência, uma melhora no tempo de execução de aproximadamente 1,8%. O WE32100 possui também modos *imediatos* para os diferentes tipos de dados, quando o dado segue o descritor de modo. Como em muitas máquinas, literais curtos e imediatos não precisam ter o mesmo tipo de dado, como os demais operandos, e esses modos não podem ser usados como destino de uma operação.

Uma instrução associa um tipo predefinido de dado com seus operandos, tal como *add word* e *add byte*. No entanto, esse tipo de dado default é essencialmente uma conveniência que provê descrições abreviadas de endereçamento. O WE32100 possui um operando de *tipo expandido* que permite especificar explicitamente o tipo do operando juntamente com a forma de endereçamento. O WE32100 executa internamente operações lógicas e aritméticas em words; a máquina efetua qualquer conversão de tipo quando da leitura ou armazenamento dos operandos.

Como exemplo de um operando de tipo expandido, considere a adição de um byte inteiro 'a' com um word inteiro 'b' e o armazenamento do resultado num halfword inteiro 'c'. Normalmente esse caso requer diversos passos de instruções. Para a maioria das máquinas, a seqüência de operações teria a forma:

```
movbw    a,temp      converte 'a' em word
addw2    b,temp      efetua a soma temporária
movwh    temp,c      converte o resultado em halfword
```

Usando-se o tipo expandido no WE32100, somente uma instrução é necessária:

```
addb3      a,{A}wordçbçhalfwordçc
```

Aqui o tipo de operando é designado entre chaves. Neste exemplo add byte foi usada como instrução para especificar o tipo do primeiro operando; se o primeiro operando também utilizasse um tipo expandido, qualquer instrução de adição produziria o mesmo resultado.

Como foi visto no exemplo acima, o tipo expandido é conveniente quando acontece de os operandos serem inconsistentes, já que ele elimina temporários que poderiam competir por registradores. O tipo expandido também provê algumas operações que não estão diretamente disponíveis com as instruções. Por exemplo, não existe operação de multiplicação sem sinal, porém essa operação pode ser conseguida com uma instrução de multiplicação utilizando operandos de tipo expandido sem sinal. Uma motivação final para isso é permitir futuras extensões de tipos.

3.3.2 OUTRAS OPERAÇÕES

Outras operações no WE32100 incluem funções para manipular strings e campos de bits. As operações com strings foram projetadas especificamente para a representação das mesmas em C, onde um string é uma seqüência de bytes terminados por um *null* (zero). Há duas primitivas para manipulação de strings:

string copy: copia um string em outro.
strind end: localiza o caractere de finalização (null) num string.

Os endereços de operandos nessas operações são especificados em registradores predefinidos. A operação *string end* pode ser usada para determinar a extensão de uma string ou, em conjunto com *string copy*, para permitir uma função para extender os bytes no final de uma string. Não há especificação de extensão em qualquer dessas operações (string copy assume que há espaço adequado como destino e string end que o string está corretamente representado). Essas instruções são adequadas para C, mas não necessariamente para outras linguagens. O WE32100 provê também uma instrução para movimentação de um bloco de dados, similar à string copy, com a diferença que a extensão é especificada.

Um *campo* (field) no WE32100 é uma seqüência de bits de extensão variável, que no entanto deve caber inteiramente dentro de uma word. Existem instruções para ler e escrever um campo, onde são especificados o número de bits, a posição do primeiro bem

como os endereços fonte e destino. Campos podem ser manipulados em termos de bytes, halfwords ou words. Usando essas operações, a maioria das manipulações de bits necessárias em linguagens de alto nível pode ser facilmente implementada.

O WE32100 dá suporte a instruções de co-processadores para aritmética de ponto flutuante e decimal. Esse suporte é dado por uma interface para co-processador que permite à CPU iniciar a instrução e ler o operando na memória. O co-processador é ativado por um comando da CPU. Ele então passa a monitorar o barramento para guardar os operandos para aquela operação. Os dados são lidos da memória pela CPU e armazenados (latched) pelo co-processador no barramento de dados.

A CPU fica bloqueada durante a execução daquela instrução pelo co-processador. Quando a mesma está terminada, o co-processador ativa um sinal (done) para avisar a CPU. As condições de estado são enviadas de volta à CPU, e o resultado é colocado no barramento de dados pelo co-processador e escrito na memória pela CPU.

3.3.3 UNIÃO DE PROCEDIMENTOS (PROCEDURE LINKAGE)

O WE32100 oferece operações de alto nível para união de procedimentos bem como um conjunto de primitivas para salto e retorno de sub-rotinas. As operações de alto nível são úteis para muitas linguagens de programação, incluindo C.

As operações de alto nível para união de procedimentos manipulam o quadro de pilha, salvam registradores e transferem o controle entre procedimentos. São implementadas para ser eficientes e incluem chamada/retorno de procedimentos e salvamento/recuperação de registradores. A instrução de push pode ser usada para armazenar argumentos. O processo de união de procedimentos manipula a pilha e execução. Quatro registradores são modificados:

PC: o contador de programa é modificado para iniciar a execução da sub-rotina e para retornar ao programa chamador.

SP: o ponteiro da pilha é ajustado convenientemente para apontar o topo da mesma.

FF: o ponteiro de quadro sinaliza o ponto da pilha exatamente acima da área de salvamento de registradores (usualmente o início do espaço para variáveis locais no procedimento).

AP: o ponteiro de argumentos sinaliza a lista de argumentos utilizados pelo procedimento. Esta lista precede os demais dados de união na pilha.

Além desses registradores, outros possuem um presumível significado semântico. Especificamente, os registradores de r0 a r2 são vistos como temporários cujos valores não são salvos entre chamadas de procedimentos. Os registradores r3 a r8 podem ser salvos por meio dessas chamadas.

Um quadro de pilha típico, como o utilizado em C, contém argumentos, informações de retorno, registradores salvos e variáveis locais. Esse quadro de pilha é mostrado na Figura 3.7.

Os quadros de pilhas que usam as operações de união de alto nível do WE32100 podem diferir do que foi colocado acima no número de registradores salvos.

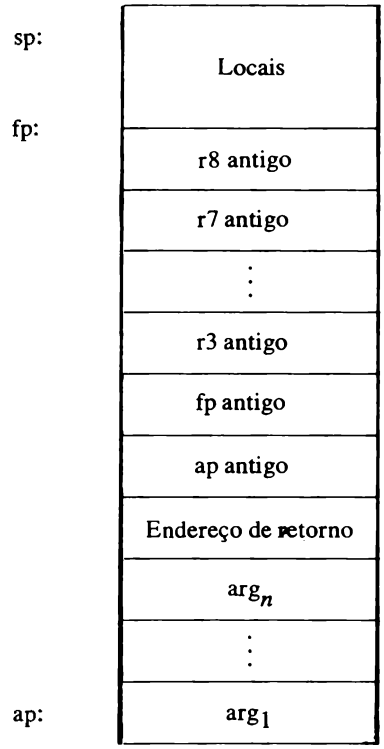


Figura 3.7 Exemplo de quadro de pilha.

Há quatro instruções usadas na união de procedimentos. O procedimento chamador usa a instrução CALL para salvar o AP e o endereço de retorno na pilha. A primeira instrução de todo procedimento é SAVE, que salva FP e uma seqüência especificada de registradores de r3 a r8. Conceitualmente só os registradores a serem usados são salvos. Como dito anteriormente, os registradores r0 a r2 não são salvos e podem ser usados para

guardar o resultado. No final, o procedimento executa a instrução RESTORE, para restaurar quaisquer valores necessários de r3 a r8, e uma instrução RETURN, que recoloca o ponteiro de quadro na posição do procedimento chamador e reassume sua execução. Os argumentos são tipicamente colocados na pilha por meio das instruções PUSH ou PUSHADDRESS. A instrução RETURN automaticamente retira-os da pilha. O uso dessas instruções permitiu um incremento de velocidade de execução de 20% sobre a codificação direta das mesmas.

Acoplados com o esquema de união de procedimentos acima, existem dois modos de endereçamento – ap-offset e fp-offset. Esses modos provêem um descritor de um byte que pode referenciar um dado cujo endereço é um deslocamento para os registradores ap ou fp numa faixa de 0 a 14. Desde que procedimentos possuam tipicamente poucos argumentos e muitas vezes poucas variáveis locais, a maioria dos argumentos e variáveis locais pode ser referenciada com um descritor de um byte. Para um exemplo de como esses modos podem ser usados, considere as variáveis no programa em C:

```
foo(a,b,c)
int a,b,c;
A
int d,e,f;
    ...
ç
```

Todas as variáveis apresentadas (a-f) poderiam ser endereçadas com um descritor de um byte. Nossa análise mostrou que cada um desses modos produziu uma redução média de espaço de 5% e uma melhora média de velocidade de execução de 1,8%.

Se for inconveniente o uso do esquema de união acima, instruções de Jump to Subroutine e Return from Subroutine são apresentadas. As instruções de Jump to Subroutine atuam como em muitas máquinas, guardando o endereço de retorno na pilha e mudando o controle de execução para a sub-rotina. A operação possui formatos para salto (que provê o endereço) e desvio (que provê um deslocamento para o pc). A instrução Return from Subroutine executa a operação inversa, porém o interessante é que ela possui uma forma condicional, isto é, a execução do retorno pode depender de códigos condicionais, sendo que todos os códigos existentes podem ser testados.

3.3.4 CONTROLE AMBIENTAL: TAREFAS E EXCEÇÕES

Fornecer funções de *sistema operacional* numa linguagem de programação, tais como controle de tarefas e exceções, tem sido tipicamente difícil de implementar. Muitos artigos sobre compiladores têm pedido assistência na arquitetura de máquinas. O WE32100 provê algo nesse sentido, particularmente com exceções.

A arquitetura orientada a processo do WE32100 facilita isso por determinar o projeto de processo com a arquitetura da máquina. Um compilador pode representar uma tarefa como um processo do WE32100 com os mapeamentos apropriados de memória que podem ser construídos tanto pelo *run time* da linguagem como pelo sistema operacional. Esse enfoque simplifica também o aspecto de mútua exclusão de encaixe de tarefas, desde que o mesmo é provido por hardware. Os outros tópicos de encaixe, sincronização e troca de dados não são diretamente assistidos por hardware e necessitam de alguma assistência do gerenciador do sistema. A Seção 3.4 sobre projeto de sistemas operacionais discute essa estrutura de processo para o WE32100.

O WE32100 possui um mecanismo de *interrupção* controlado por tabela que é usado para gerenciar exceções do sistema (e outras chamadas do sistema) e pode servir também para implementar exceções do usuário. Quando ocorre uma exceção a nível de sistema o WE32100 efetivamente executa uma instrução de call com transferência controlada usando um conjunto predefinido de valores de operandos. Essa operação de transferência é melhor visualizada como uma forma de *jump-to-subroutine*, onde tabelas selecionam o endereço apropriado da sub-rotina. Se o sistema operacional possui uma interface para modificar as tabelas de *transferência controlada*, um tratamento de exceção escrito por usuário pode ser chamado automaticamente colocando-se o seu endereço no lugar apropriado da tabela. As exceções definidas pelo usuário podem ser manejadas por essa operação, adicionando-se endereços de tratamento de exceção para cada exceção de usuário e fazendo com que o mesmo execute uma operação de chamada com transferência controlada quando a exceção ocorre. A transferência controlada pode transferir para uma rotina normal do usuário. Para reassumir-se a execução após a exceção, deve-se apenas executar uma instrução de retorno com transferência controlada.

3.3.4.1 BIBLIOTECAS COMUNS E PACOTES

Uma forma de implementar-se bibliotecas comuns e pacotes (tipos de dados abstratos) é a utilização do processo de transferência discutido acima. Conceitualmente, a chamada a um pacote pode ser vista como uma exceção de usuário.

Uma (ou mais) das tabelas de transferência poderia ser alocada para controle de pacotes. Cada entrada nessa tabela corresponderia a uma entrada de função em algum pacote. Para acessar-se um procedimento de pacote, uma instrução de chamada com transferência controlada seria usada, e cada procedimento de um pacote retornaria numa operação de retorno com transferência controlada. Uma grande vantagem desse sistema é que o código para procedimentos de pacote poderia ser dividido através de processos, e a

geração de código para chamada de pacotes também poderia ser simplificada.

3.4 SUPORTE AO SISTEMA OPERACIONAL DO WE32100

O WE32100 foi projetado para prover um ambiente eficiente para um sistema operacional sofisticado. Não há sistema operacional construído dentro do processador, nem é o mesmo otimizado para um sistema operacional particular. Em vez disso, o processador provê dois mecanismos que podem ser usados para manipular processos, transferências controladas para o sistema operacional e responder a interrupções e manejar exceções.

3.4.1 PROCESSOS

O WE32100 dá suporte a um sistema operacional orientado a processo; um modelo particular de processo é implícito na arquitetura da máquina. Esse modelo possui diversas características:

- há quatro níveis de execução privilegiada para permitir flexibilidade na construção de sistemas operacionais multinível. A hierarquia entre os quatro níveis é dada apenas pelo mecanismo de transferência controlada;
- há apenas uma pilha de execução por processo. Essa pilha é usada pelo mecanismo de chamada do procedimento bem como pelo mecanismo de transferência controlada e é usada independentemente do nível de execução;
- um processo é definido para o processador por *um bloco de controle do processador* (processor control block – PCB), que armazena cópias dos recursos do processador usados pelo processo (por exemplo, os registradores internos);
- pretende-se que pelo menos o núcleo do sistema operacional resida no espaço de endereçamento de todos os processos.

Com quatro níveis de execução, um sistema operacional que necessitasse de uma pilha separada para cada nível teria de manter pelo menos quatro segmentos de pilha crescentes. Com uma única pilha, o sistema operacional necessita manter apenas uma, e um mecanismo de overflow para uma única pilha é suficiente para incrementar a pilha. O mecanismo de erros de pilha, que gerencia condições de overflow bem como outras violações, é descrito na Seção 3.4.6.2. Além disso, quatro registradores seriam necessários

para apontar essas quatro pilhas, e o gerenciamento de tais registradores especiais é caro num projeto VLSI. A pilha de execução pode também ser usada para passar argumentos do código do usuário para funções do sistema; o mecanismo regular de passagem de parâmetros pode ser usado sem elaboradas operações de cópia.

No núcleo do sistema operacional está o espaço de endereçamento de todos os processos, eliminando a necessidade de cópia de buffers do usuário para buffers do sistema, já que o sistema pode acessar buffers do usuário e vice-versa. Um espaço comum de endereçamento para usuário e sistema operacional é necessário para que o mecanismo de pilha única de execução possa funcionar: a troca de espaço de endereçamento perderia a pilha e a corrente de procedimentos que ela contém. O mecanismo de exceção do WE32100 espera que o núcleo esteja no espaço de endereçamento de todos os processos, de tal forma que o processador não mude o gerenciamento da memória para acessar um tratamento de exceção.

Duas estruturas de dados estão associadas com processos no WE32100: o PCB e a pilha de interrupção. O PCB (veja Figura 3.8) possui espaço para os quatorze registradores utilizados por um processo. Esses são os onze registradores de usuários mais três de controle, o *stack pointer* (SP), o *program counter* (PC) e o *processor status word* (PSW). Duas palavras no PCB são usadas para guardar os limites inferior e superior de endereço da pilha de execução; esses limites são checados no mecanismo de transferência controlada. O resto do PCB não possui limite de extensão e é previsto para ser usado (mas não restrito) pelo gerenciamento de memória. A pilha de interrupção não está associada a nenhum processo e contém ponteiros ao PCB. Um dos registradores internos não associados com qualquer processo, o ponteiro ao PCB (PCBP), aponta o PCB do processo corrente no processador. O segundo desses registradores, o *interrupt stack pointer* (ISP), aponta o topo da pilha de interrupção. Ambos, o PCBP e o ISP, são privilegiados no sentido de que podem ser escritos apenas quando o processador está no nível de execução núcleo.

3.4.2 CHAVEAMENTO DE PROCESSOS

O primeiro dos dois mecanismos usados para suporte a sistema operacional é o de *chaveamento de processos,* que é utilizado no chaveamento de processos, tratamento de interrupções e exceções. Esse mecanismo possui quatro partes que são usadas pelas microsseqüências em várias combinações:

1. Guarda os registradores de controle no PCB apontado pelo PCBP (o PCB "velho"). Guarda os registradores do usuário no PCB "velho" (opcional).

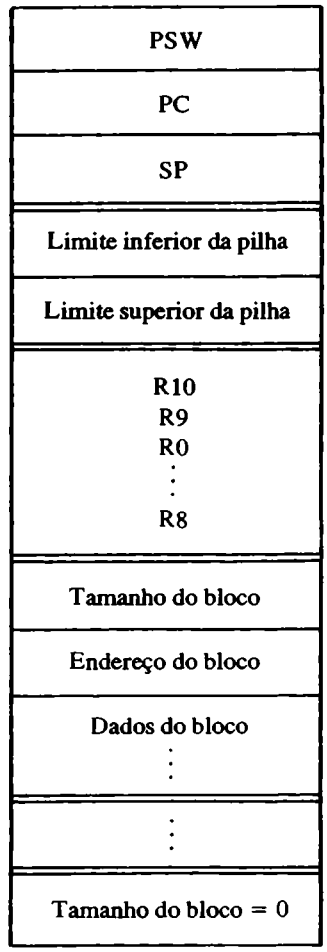


Figura 3.8 Diagrama do bloco de controle de processos.

2. Atualiza o PCBP para apontar o "novo" PCB. Carrega os registradores de controle do "novo" PCB. Move o PCBP além do contexto inicial do "novo" PCB (opcional).

3. Executa uma série de movimentações de bloco (opcional).

4. Carrega registradores do usuário do "novo" PCB (opcional).

Os dados na seção de movimentação de bloco do PCB são previstos para especificação de mapeamento de memória. Desde que toda E/S no WE32100 é mapeada em memória, o endereço inicial na seção de movimentação de bloco seria a base de tradução de registradores numa unidade de gerenciamento de memória. Com esse mecanismo, o

chaveamento de processo estabeleceria automaticamente o domínio de endereçamento virtual do novo processo sem mais nenhuma intervenção do sistema operacional. Naturalmente, se o mecanismo do WE32100 é indesejável para alguma aplicação, ele pode ser desabilitado colocando-se zero no contador de movimentação de bloco em todos os PCB.

3.4.3 CHAMADA E RETORNO DE PROCESSO

Instruções explícitas são providas no WE32100 para chaveamento de processos pelo sistema operacional. Elas não são usadas para a organização de processos, que no WE32100 ainda está a critério do software do sistema operacional. Em vez disso, elas provêem meios para o despacho de processos e coordenação de chaveamento de processos determinados pelo sistema operacional com aqueles que aparecem inesperadamente de interrupções. As duas instruções, Call Process e Return to Process, são análogas ao par Jump to Subroutine e Return from Subroutine. Nas instruções de transferência para sub-rotina o endereço inicial define a sub-rotina. O "jump" guarda o endereço de retorno na pilha de execução e o "return" o retira. Nas instruções de transferência para processo, o endereço do PCB define o processo. Call guarda o endereço do PCB corrente na pilha de interrupção e o "return" o retira. Como nas instruções de transferência para sub-rotina, as de transferência para processo apenas transferem o fluxo de controle e não passam argumentos explicitamente.

A instrução Call Process tem o endereço do PCB como seu argumento. Ela salva o contexto do processo anterior no PCB anterior, com o PC salvo apontando para a próxima instrução a ser executada, e retira um novo contexto do novo PCB. A instrução Return to Process apenas carrega um novo contexto do novo PCB.

3.4.4 INTERRUPÇÕES

O mecanismo de interrupção do WE32100 foi concebido para ser eficiente, confiável e consistente com o modelo de processo do processador. Desde que interrupções são assíncronas, elas provavelmente não estão associadas a nenhum processo corrente no processador. Idealmente uma interrupção deveria ser tratada por um novo processo, que é exatamente o que o processador faz. Um processo de interrupção possui um contexto totalmente novo e é improvável que interfira com qualquer outro processo. Se o dispositivo gerador da interrupção não é um recurso crítico, a rotina de tratamento não precisa rodar em modo sistema, mas pode ser despachada diretamente em modo usuário. Uma pilha especial de execução para interrupções não é necessária, já que cada processo de inter-

rupção recebe uma nova pilha de execução que não necessita de nenhum tratamento especial.

3.4.4.1 MECANISMO DE INTERRUPÇÃO

Uma interrupção no WE32100 é tratada como uma instrução Call Process inesperada. O dispositivo que interrompe apresenta uma identificação de 8 bits. Essa identificação seleciona um entre 256 PCB numa tabela começando num endereço virtual fixo. Cada ponteiro a um PCB corresponde a um processo de tratamento de interrupção. A microsseqüência é então exatamente como na instrução Call Process. O processo de interrupção então rodará, a menos que seja interrompido por outro de maior prioridade. Quando o processo termina, deve-se dar uma instrução de Return to Process, que reiniciará o processo interrompido.

A pilha de interrupção monitoriza as interrupções encadeadas. A menos que a pilha de interrupção seja explicitamente manipulada pelo sistema operacional, o PCBP no fim da pilha aponta os PCB de processos de tratamento de interrupção de prioridades crescentes, com o PCBP apontando o PCB da interrupção de mais alta prioridade sendo atualmente executada.

3.4.5 TRANSFERÊNCIA CONTROLADA

Esse mecanismo provê um meio de entrada controlada num procedimento ou rotina com um novo PSW e pode ser usado como um mecanismo de chamada de sistema. Consiste em uma chamada controlada e um retorno controlado. A chamada controlada opera como um Jump to Subroutine, exceto que o par PC/PSW é guardado e trocado. Essa instrução possui dois operandos que funcionam como um índice a uma tabela dupla para determinar o novo PSW e o endereço a pular. O par PC/PSW é retirado da pilha no retorno controlado.

Numa posição predefinida da memória há uma tabela de ponteiros de "primeiro nível", sendo que cada um deles pode apontar uma tabela de "segundo nível" de pares PC/PSW. O primeiro operando de índice seleciona a tabela apropriada de "segundo nível". O segundo índice determina o endereço a pular (veja Figura 3.9). O "primeiro nível" permite 32 entradas e cada tabela de "segundo nível" permite até 4095 entradas. a Tabela de "segundo nível" pode estar localizada em qualquer ponto da memória. Em particular, elas podem ser divididas por alguns (ou todos) usuários ou serem únicas num processo.

3.4.5.1 USO DE TRANSFERÊNCIA CONTROLADA

Em arquiteturas mais convencionais existe apenas uma instrução Call Supervisor que carrega um novo PC e PSW de uma posição predefinida. Fica por conta do sistema operacional a determinação de qual procedimento chamar.

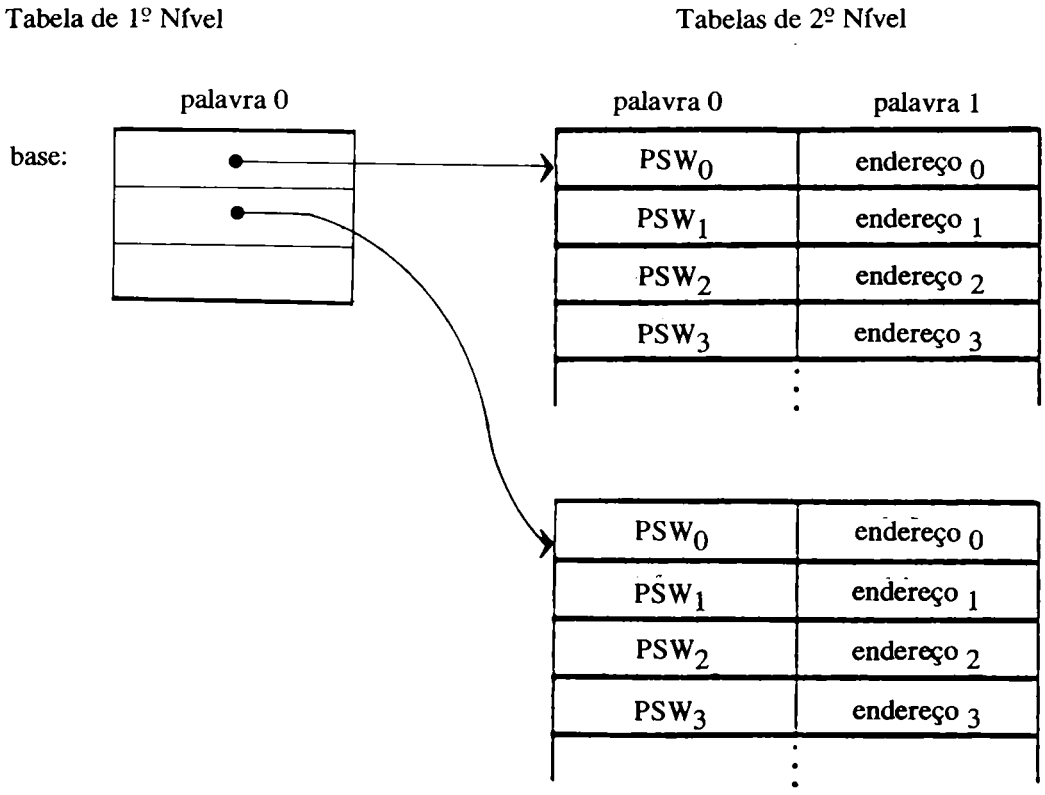


Figura 3.9 Tabelas de transferência controlada

No WE32100 esse procedimento de software é assistido pela arquitetura interna. O mecanismo de transferência controlada é o único meio pelo qual um processador pode mudar seu nível de execução. Não há noção de nível de execução na microsseqüência de chamada, pois o novo PSW determinará o nível de execução do processo. A instrução de retorno controlado é o único ponto onde o processador reconhece privilégio: ele não permite que um procedimento retorne a outro mais privilegiado.

Desde que há apenas uma pilha de execução para cada processo, é extremamente importante manter a integridade da mesma. Isso é feito na transferência controlada. Há duas entradas no PCB, que correspondem aos limites superior e inferior da pilha de

execução. A instrução de chamada controlada verifica se o ponteiro à pilha encontra-se nesses limites antes de executar a transferência. O sistema operacional tem de manter corretamente os limites de pilha no PCB.

3.4.6 EXCEÇÕES

Exceções são eventos que indicam que algo está errado com a execução corrente. Podem ser detectadas internamente pelo processador ou ser geradas externamente. O WE32100 pode tratar todos os tipos de exceção sem parar, usando os dois mecanismos básicos de transferência controlada e chaveamento de processos. Existem quatro níveis de exceções no WE32100, dependendo da gravidade do erro e dos recursos disponíveis no momento. São eles: exceção normal, exceção de pilha, exceção de processo e exceçao de reset.

3.4.6.1 EXCEÇÃO NORMAL

A maioria das exceções é do tipo normal, incluindo-se aí instruções ilegais, overflow e acesso a registradores privilegiados, bem como erros gerados externamente. O processador guarda um índice de exceções num campo de 4 bits do PSW chamado *código de estado interno* (Internal State Code – ISC). A ação do processador é idêntica à transferência controlada, com o índice de primeiro nível sendo zero e o de segundo nível, ISC. Portanto, na ocorrência de uma exceção normal, o processador transfere o controle automaticamente para uma rotina de tratamento, que não precisa ser privilegiada ou mesmo parte do sistema operacional. Caso o usuário possua a rotina de tratamento, basta mudar-se a entrada da tabela para aquela exceção para apontar essa rotina.

3.4.6.2 EXCEÇÃO DE PILHA

A pilha de execução é um recurso crítico e, portanto, tem de ser mantida cuidadosamente. Caso haja problemas com a pilha, uma exceção normal não pode ser tratada, desde que o tratamento começa sua execução guardando o par PC/PSW na pilha. Portanto existe um mecanismo especial de chaveamento de processo para esse caso. Essas exceções são detectadas durante uma transferência controlada, tanto quando o teste de limites de pilha acusa erro, como quando uma leitura ou escrita na pilha não é bem sucedida. Quando isso ocorre, o processador carrega um ponteiro para um novo PCB de uma posição predefinida, guarda o PCBP corrente na pilha de interrupção, salva o conteúdo dos registrado-

res de controle e carrega um novo conjunto, obtendo uma nova pilha. Ele não executa as movimentações de bloco, de tal forma que o gerenciamento de memória não é forçado a mudar. O novo processo pode consertar a pilha, ajustar seus limites, matar o processo com problema ou o que for mais conveniente.

3.4.6.3 EXCEÇÃO DE PROCESSO

Uma exceção de processo ocorre quando de um erro de memória durante uma leitura ou escrita no PCB em uma microsseqüência. Já que há problemas no PCB, o processo está efetivamente morto e não há como salvar registradores ou recomeçá-lo. Tudo que o processador pode fazer é iniciar um novo processo. No caso dessa exceção, o processador carrega um ponteiro a um novo PCB de uma posição virtual predefinida, guarda o PCBP corrente na pilha de interrupção e carrega um novo conjunto de registradores de controle. Como o tratamento desse tipo de exceção deve ser breve, é conveniente incluí-lo no domínio de todos os processos, evitando assim as movimentações de bloco do chaveamento de processos.

3.4.6.4 EXCEÇÃO DE RESET

Essa exceção ocorre quando tudo o mais falha ou quando o processador recebe reset externo. Ocorre quando de um erro de memória durante uma leitura ou escrita na pilha de interrupção, durante a leitura de um vetor de endereço ou durante o processamento de uma exceção de processo. Quando ela ocorre, o processador desabilita o endereçamento virtual, carrega o ponteiro a um novo PCB de uma posição física predefinida e carrega um novo conjunto de registradores de controle. Caso ocorra um erro de memória durante o processamento de exceção de reset uma outra exceção de reset é gerada e o processador tenta novamente. O microprocessador é totalmente reiniciável em qualquer condição de erro.

3.4.7. USO DO CHAVEAMENTO DE PROCESSOS POR HARDWARE E DA MMU PELO UNIX

As características de chaveamento de processo da CPU e as de memória virtual da MMU combinam-se para dar suporte a operações típicas de sistemas operacionais modernos. Exporemos aqui o uso dessas características na implementação de chaveamento entre processos de usuário.

A MMU trabalha com o conceito de seções de endereçamento virtual, onde cada seção consiste em um quarto do espaço de endereçamento de 32 bits e é mapeada através de uma tabela de descritores de segmento. A MMU pode descarregar automaticamente seus caches internos de descritores numa base por seção em resposta a uma única escrita da CPU. Isso significa que em uma operação o sistema operacional pode alterar o mapeamento de um quarto do espaço de endereçamento virtual e que a MMU garantirá que as porções apropriadas de seu cache interno serão descarregadas.

Uma maneira natural de dividir a memória virtual é dedicar uma ou mais seções ao sistema operacional e as restantes aos processos usuários correntemente ativos. Isto é, todos os processos usuários residirão nos mesmos endereços virtuais (é claro que apenas um processo pode residir aí por vez). Reescrevendo os registradores do MMU que contêm as tabelas de descritores de segmento para essas seções, o sistema operacional pode chavear processos usuários no espaço virtual com muitas operações.

De fato, o código completo do sistema operacional para o chaveamento de processos usuário seria o seguinte:

```
{Correntemente rodando o processo usuário A}

/*chaveamento para o chaveador de processos do sistema operacional*/

MOVAW switcher,%r0
CALLPS

{Agora rodando no chaveador de processos do sistema operacional}

/*reescreve os registradores da MMU para duas seções*/

MOVW newpointer1,mmureg1
MOVW newpointer2,mmureg2

/*chaveamento de volta para o novo processo B*/

RETPS

{Agora rodando o processo B}
```

Esse código efetua um chaveamento completo entre processos usuários. A instrução CALLPS (Call Process) salvou o conteúdo dos registradores do processo corrente (processo usuário A) e obteve os registradores de um processo do sistema operacional denominado *processo chaveador*. O processo chaveador, após determinar qual processo

usuário para o qual chavear (não mostrado), escreveu os endereços desses processos nas tabelas de descritores da MMU, fazendo o processo A "desaparecer" do espaço de endereçamento virtual e o processo B "aparecer" aí. Finalmente, foi executada uma instrução RETPS (Return to Process), que obteve o conteúdo dos registradores do processo B tal como foram deixados da última vez que esse processo executou uma CALLPS para o processo chaveador.

Prover suporte a sistemas operacionais é vantajoso, visto que as primitivas são garantidamente corretas, protegidas de alteração ou destruição e podem acessar facilidades internas da CPU para desabilitar interrupções, iniciar seqüências de erros e assim por diante.

3.5 USO DE PASTILHAS PERIFÉRICAS COMUNS COM OS SISTEMAS WE32100

O conjunto de pastilhas WE32100 provê uma interface pseudosíncrona para o projetista de sistema. O protocolo dessa interface é compatível com periféricos comuns, tais como UART, controladores de interrupção etc. O conjunto WE32100 permite interface fácil com memórias dinâmicas, estáticas e projetos de cache.

A transação de memória básica é de 3-7 ciclos de relógio (nenhum estado de espera) com um ciclo adicional usado pela MMU WE32101 para prover um ambiente de memória virtual quando a mesma é utilizada.

Algumas características exclusivas do conjunto de pastilhas são:

1. Uma aquisição de bloco dinamicamente selecionável que melhora o desempenho da CPU eliminando traduções triviais de endereços virtuais.
2. Exceções de barramento retardadas permitindo ao projetista um tempo maior para determinar falhas.
3. Arbitração de barramento com 2 sinais.

3.5.1 INTERFACE COM CO-PROCESSADOR

A interface com co-processador suportada pelo WE32100 provê uma interface de alta velocidade entre a CPU e a MAU, juntamente com a flexibilidade para expansões com co-processadores adicionais. Um total de dez códigos de CPU foram reservados ao co-processador, provendo uma variedade de operações com zero, um e dois operandos,

com cada operando podendo ter extensão de uma, duas ou três palavras. A comunicação entre a CPU e co-processador tem a forma de uma transação mínima de barramento, com estados de espera opcionais. Dados recolhidos escritos na memória são acessados concorrentemente pela CPU e o co-processador numa única transação de barramento em vez de serem roteados através da CPU em duas transações. Essa forma provê a máxima velocidade possível de transferência de operandos.

Uma operação com co-processador é iniciada com uma transação de *transmissão de co-processador*, na qual um campo de identificação de co-processador de 8 bits permite o endereçamento de até 256 tipos diferentes de co-processador no barramento. Os 24 bits restantes da palavra de transação podem ser interpretados pelo co-processador como este bem queira, usualmente como uma informação de código de instrução e operando. Todos os operandos são então recolhidos e tomados diretamente pelo co-processador. Nesse ponto o co-processador executa a operação requisitada e indica o final à CPU com um sinal DONE. Esta armazena então o estado do co-processador no PSW e provê os endereços para qualquer escrita na memória que possa ser requerida pela operação do co-processador. Quaisquer exceções que possam ter sido detectadas pelo co-processador podem ser sinalizadas à CPU durante o ciclo de transferência de estado.

3.6 ARQUITETURA DA UNIDADE DE GERENCIAMENTO DE MEMÓRIA WE32101

A WE32101 é uma unidade de gerenciamento de memória de segunda geração[3]. A primeira geração foi implementada em tecnologia CMOS de 2,5 μm e roda a 7,2 MHz. A WE32101 é implementada em tecnologia CMOS de 1,5 μm com relógio de 18 MHz. Ela provê tradução de endereços para segmentos contíguos e paginados simultaneamente. Sua arquitetura possui os seguintes atributos:

- facilita a organização sistemática de memória para sistemas operacionais, dividindo endereços virtuais em unidades manejáveis de seções, segmentos e páginas;
- teste de proteção de acesso para segmentos contíguos e paginados;
- suporte a algoritmos de paginação e segmentação por atualização automática de bits "referenciados" e "modificados";
- detecção automática de erro de processamento para segmentos internos e paginados de entrada de descritores no cache;
- suporte a segmentos de tamanhos variáveis de 8 bytes a 128 kbytes;

- suporte a segmentos compartilhados através do uso de descritores indiretos de segmentos;
- facilidades de descarregamento de cache de descritores;
- detecção e resolução detalhada de erros.

Para minimizar o tempo de tradução da memória, a MMU possui um cache de descritores internos. Há um cache de 32 descritores de segmento, diretamente mapeados, e um cache de 64 descritores de página, associativos e bidirecionais.

Para traduzir um endereço a MMU procura seus caches de descritores em busca de descritores relevantes. Se estes estão presentes, a MMU testa se houve violação de extensão ou de permissão de acesso.

Para segmentos contíguos, a tradução é feita adicionando-se o endereço de base, do cache de descritores, a um deslocamento do endereço virtual, para formar o endereço físico.

Para segmentos paginados, a MMU concatena um endereço de base de página, do descritor de página do cache, ao deslocamento de página, do endereço virtual, para formar o endereço físico.

Se houver uma falha no cache, a MMU executará um "processamento de falha". Ela acessará as tabelas de descritores na memória e as carregará nos caches internos.

3.6.1 DIVISÃO DE ESPAÇO DE ENDEREÇAMENTO

A MMU suporta até 2^{32} bytes de memória física ou virtual. A tradução de endereços virtuais pode ser selecionada dinamicamente pela CPU através de uma linha virtual/físico.

O espaço de endereçamento virtual é dividido em seções e estas, em segmentos. Segmentos podem ser paginados ou contíguos (não paginados).

O espaço de endereçamento virtual de um processo é dividido em quatro seções. Um propósito importante das seções é otimizar o descarregamento de entradas de cache de espaço de endereçamento virtual compartilhado. Os descritores em cache de uma seção podem ser descarregados independentemente de outras seções durante o processo de chaveamento. Portanto seções de espaço de endereçamento virtual (tais como bibliotecas de sistema ou rotinas de núcleo) que são comuns a múltiplos processos podem ser mantidas nos caches de descritores da MMU durante chaveamento de processos.

Cada seção pode consistir em segmentos de até 8 K. Identificadores de seção armazenadas na MMU designam as tabelas de descritores de segmentos associadas com as seções. Reescrever um identificador de seção causa descarga automática das correspondentes entradas nos caches de descritores para aquela seção.

Segmentos podem ser contíguos ou paginados. Ambos são suportados simultaneamente pela MMU. Um segmento paginado é dividido em páginas de 2 kbytes.

3.6.2 CARACTERÍSTICAS

3.6.2.1 CAMPOS DE ENDEREÇAMENTO VIRTUAL

A subdivisão do espaço de endereçamento virtual requer a divisão do endereçamento virtual em três campos para segmentos contíguos e quatro no caso de paginados.

Um endereço virtual que referencia um segmento contíguo é dividido num campo de identificação de *seção* (SID) (qual seção do espaço de endereçamento), um de *seleção de segmento* (SSL) (qual segmento dentro da seção) e um de *deslocamento* (offset) *no segmento* (SOT) (qual byte dentro do segmento) (veja Figura 3.10).

Para segmentos paginados o campo SOT é subdividido num campo *selecionador de página* (PSL) (qual página no segmento) e um *deslocamento na página* (qual byte dentro da página) (veja Figura 3.11).

3.6.2.2 DESCRITORES E TABELAS DE MAPEAMENTO

Tabelas de descritores de segmentos (SDT) e de páginas (PDT) especificam mapeamentos entre espaços de endereçamento físico e virtual em termos de segmentos e páginas. As tabelas precisam residir em memória física sempre que a MMU necessitá-las. Um segmento contíguo é representado por apenas uma entrada na SDT, enquanto um segmento paginado é representado por uma entrada na SDT e uma na PDT. No segundo caso a entrada na SDT contém o endereço de base físico da PDT.

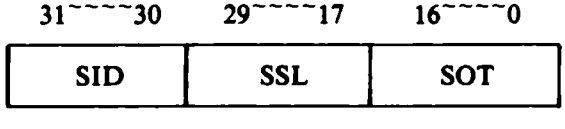


Figura 3.10 Campos de endereço virtual, para um segmento contíguo.

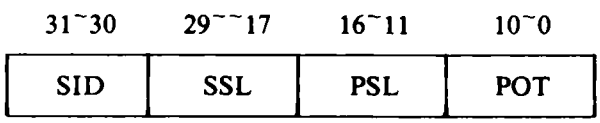


Figura 3.11 Campos de endereço virtual para um segmento paginado.

A MMU guarda ponteiros para as tabelas em memória e também contém o comprimento dessas tabelas. Essa informação é usada quando da correção de processamento.

3.6.2.3 DESCRITORES E TABELAS DE SEGMENTOS

O mapeamento de endereços virtuais em físicos para uma seção é definido pela SDT a ela associada. Uma SDT contém uma entrada de 8 bits, um descritor de segmento para cada segmento na seção. Cada segmento pode consistir em até 128 kbytes. O campo de seleção de segmento (SSL) do endereçamento virtual é usado como índice para a SDT (veja Figuras 3.12 e 3.13).

V~(Válido) V = = 1 indica um descritor de segmento válido. Se V = = 0, um acesso causa um erro.

C~(Contíguo) C = = 0 significa que o segmento é paginado e C = = 1, que é contíguo. Se C = = 0, a MMU usa a segunda palavra do SD como endereço físico de uma PDT. Se C = = 1, a MMU usa a segunda palavra do SD como endereço de base físico de um segmento contíguo.

P~(Presente) P = = 1 significa que o segmento ou a PDFT estão presentes na memória. Se a MMU tenta usar o SD durante uma correção de processamento e P = = 0, ocorre um erro.

I~(Indireto) I = = 1 indica um descritor de segmento indireto (que aponta para outro descritor). Durante a correção de processamento, se I = = 1, a segunda palavra do SD é usada como endereço físico de outro SD e o segundo SD é lido. Se I = = 1 no segundo SD, o procedimento é repetido. Podem ocorrer até seis leituras de SD, mas se no sexto SD I = = 1, ocorre um erro. Descritores indiretos de segmento podem ser usados para implementação de segmentos compartilhados que podem ser facilmente trocados. O único descritor de segmento que tem de ser mudado pelo sistema operacional quando dessa troca ou movimentação é o último (aquele com I = = 0).

31~24	23~~~10	9~8	7	6	5	4	3	2	1	0
Acc	Max~ Off	Res	I	V	R	T	$	C	M	P

Figura 3.12 Formato da primeira palavra de um SD.

31~~~3	2~~0
Endereço (alta ordem 27 ou 29 bits)	Soft

Figura 3.13 Formato da segunda palavra de um SD.

R (Refereciado) Se a MMU está configurada para tanto e um descritor de segmento (numa tabela de descritores) é referenciado via uma correção de processamento e a referência não está com erro, o bit R do descritor é sinalizado em 1.

M (Modificado) Se a MMU está configurada dessa forma ela sinaliza o bit M em um, caso ele já não esteja sinalizado, se ela traduz com sucesso um endereço usando esse SD para um acesso de escrita.

$ (Cachable) Sempre que o SD é usado para uma tradução, a MMU sinaliza seu pino de cache para refletir o valor desse bit. Esses bits podem ser usados pelo sistema para prevenir ou autorizar o armazenamento em cache de instruções e dados numa base por segmento.

T (Trap de objeto) Se há uma tradução válida usando o descritor de segmento (permissões de acesso e máximo deslocamento – offset – não foram violadas) e o segmento é contíguo (C==1) e o bit T está em um, ocorre um erro de trap de objeto. Esse bit é ignorado em segmentos paginados.

Res (Reservado) Esses bits são reservados para uso futuro da MMU.

Soft (Reservado para Software) Esses bits são reservados para uso pelo software e garantidamente não são alterados pela MMU.

Max Off (Deslocamento Máximo) Esse campo é usado para calcular o deslocamento máximo do início do segmento que um endereço virtual pode especificar. Se um endereço virtual especifica um valor acima desse, um erro é sinalizado para a CPU.

Esse valor é igual ao número de palavras duplas (8 bytes) num segmento, menos um. Portanto é impossível especificar um segmento contendo 0 bytes com esse mecanismo.

Sempre que um descritor é usado para executar uma tradução, a MMU testa o campo de permissões de acesso do SD, o tipo de acesso sendo requisitado pela CPU e o nível de execução no qual está havendo essa requisição. Se a MMU determina que o acesso não é permitido sob essas condições, ocorre um erro.

A MMU usa o nível de execução (núcleo, executivo, supervisor e usuário) no qual o acesso está sendo requisitado e o conteúdo do campo de permissões de acesso para determinar se nesse nível é permitido leitura/escrita (RW), leitura/execução (RE), apenas leitura (RO) ou nenhum acesso (NA) no segmento. Ela verifica então o tipo de acesso requisitado pela CPU para determinar quais permissões são necessárias para permitir o acesso. Se essas permissões não são satisfeitas, ocorre um erro. Senão, o acesso é permitido.

O campo de permissões de acesso possui 8 bits e é estruturado em campos de 2 bits, um para cada nível de execução. O tipo de acesso permitido num determinado nível é codificado nesses campos de 2 bits.

O campo de endereçamento da segunda palavra de um descritor de segmento numa SDT pode ser usado pela MMU num dos seguintes modos:

1. Como endereço físico de um segmento contíguo. Esse é o endereço em memória física onde o conteúdo do segmento começa.

2. Como um endereço de PDT para um segmento paginado. Esse é o endereço em memória física onde começam as tabelas de descritores de página.

3. Como endereço físico de outro SD no caso de um SD indireto.

3.6.2.4 TABELAS E DESCRITORES DE PÁGINAS

Para um segmento paginado, a segunda palavra do descritor é usada pela MMU como endereço de uma tabela de descritores de página. Uma PDT contém descritor de página de 4 bytes para cada página no segmento. Uma PDT pode consistir em um máximo de 64 descritores de página, mas pode conter menos (veja Figura 3.14).

P~(Presente) P==1 se a página está presente; P==0, se não. Se a MMU tenta ler um PD durante uma correção de processamento e P==0, o restante das informações do PD é ignorado e ocorre um erro.

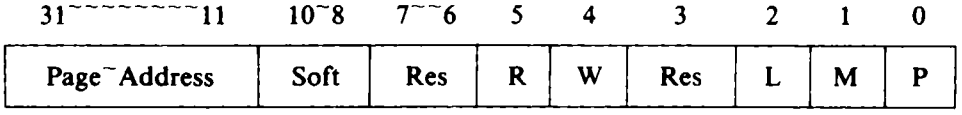


Figura 3.14 Formato de descritor de página.

R ~ (Referenciado) A MMU sinaliza R em um quando traduz com sucesso um endereço para qualquer tipo de acesso.

M ~ (Modificado) A MMU sinaliza M em um, se já não estiver nesse estado, quando traduz com sucesso um endereço para um acesso de escrita.

L~(Último) Durante a tradução, se L==1 a MMU acessará o campo de máximo deslocamento do SD correspondente e executará uma verificação de extensão. Se L==0, a tradução continua sem essa execução.

Espera-se que L==1 se o descritor de página corresponde à última página de um segmento cuja extensão não é um múltiplo integral de 2 kbytes. Se L==0, a extensão da página é 2 kbytes.

W~(Erro na Escrita) Se permissões de acesso e extensão não foram violadas e o tipo de acesso é de escrita e W==1, ocorre um erro. Esse bit pode ser usado para minimizar a cópia de páginas num processo de duplicação. Em vez de copiar todas as páginas, cada uma pode ser compartilhada tendo W==1. Uma tentativa de escrever numa página compartilhada por parte de qualquer dos processos causará um erro, e somente então aquela página terá de ser duplicada.

Page~Address (Page~Address)*2k é o endereço do primeiro byte da página em memória física.

3.6.3 ESTRUTURA INTERNA

As seguintes entidades na MMU podem ser acessadas no modo mapeado em memória:

1. O cache de descritores de segmento (32 descritores; mapeado diretamente; total de 64 palavras).

2. O conjunto bidirecional associativo de cache de descritores de página (64 descritores; conjunto associativo bidirecional; total de 128 palavras).

3. A seção RAM A. A SRAMA contém os endereços das SDT e pode ser usada para descarregar seções dos caches de descritores.

4. A seção RAM B. A SRAMB contém o número de entradas em cada SDT.

5. O registrador de erros de código. O FLTCR é usado para armazenar informações sobre erros detectados pela MMU.

6. O registrador de erros de endereço. O FLTAR contém o endereço virtual que estava sendo processado quando da ocorrência de um erro.

7. O registrador de configuração. O CR contém bits que especificam aspectos configuráveis do comportamento da MMU.

8. O registrador de endereço virtual. O VAR contém o endereço virtual a ser traduzido pela MMU e pode ser usado para descarregar entradas simples dos caches descritores.

3.6.3.1 CACHES DE DESCRITORES

O cache de descritores de segmento consiste em 32 descritores de 64 bits que diferem dos SD nas SDT que foram descritas anteriormente quanto ao formato. SD em cache contêm um campo de identificação e não contêm os bits V, I, P e R, entre outras diferenças.

O SDC é um cache diretamente mapeado dividido em quatro partes que correspondem às quatro seções. Ele necessita um índice de 5 bits para selecionar uma entrada e uma identificação de 10 bits para individualizar o descritor.

Os caches de descritores de página consistem em 64 descritores de 64 bits organizados num conjunto associativo bidirecional. O formato dos PD em cache é diferente dos PD em PDT em memória. Eles contêm um campo de identificação e um de permissões de acesso (copiados do SD) e não contém os bits P e os de software, entre outras diferenças.

O PDC é dividido em quatro partes que correspondem às quatro seções. Como no SDC, o índice é de 5 bits. A identificação, no entanto, é de 16 bits.

3.6.3.2 OUTROS OBJETOS

A seção RAM A consiste em quatro palavras de 32 bits. Cada uma delas é usada como endereço de base físico de uma tabela de descritores de segmento (SDT) de uma seção durante a correção de processamento.

A seção RAM B contém o número de segmentos em cada seção menos um. Isso é usado para testes de fronteiras em tabelas de descritores de segmento durante a correção de processamento. A SRAMB consiste em quatro palavras de 32 bits.

O registrador de endereço virtual contém o endereço virtual a ser traduzido pela MMU.

O registrador de erros de código mantém um histórico de erros e estados operacionais na pastilha de MMU. Seu conteúdo é reescrito toda vez que ocorre um erro.

O FLTAR contém o endereço virtual que estava sendo processado quando da ocorrência do último erro. O conteúdo do FLTAR é modificado quando a CPU escreve nele em modo periférico e quando ocorre um erro.

O registrador de configuração possui 32 bits e contém três itens de um bit chamado $, Ref e Mod. Os bits Ref e Mod habilitam ou desabilitam a sinalização dos bits de segmento R e M na memória. Durante a correção de processamento e atualização de R e M, o bit $ determina o estado de um pino de saída. Este pode ser ligado a um cache externo para prevenir ou impedir o armazenamento de descritores nesse cache.

3.6.4 OPERAÇÕES

Esta seção descreve diversas operações que a MMU executa em resposta a ações da CPU. A maioria das operações é bastante complexa, mas mesmo assim elas são executadas sem qualquer intervenção da CPU.

3.6.4.1 TRADUÇÃO, CORREÇÃO DE PROCESSAMENTO E ATUALIZAÇÃO DE R E M

Tradução é a operação durante a qual os caches são acessados para determinar a existência de certos descritores (aqueles que são necessários). Se esses descritores não estão presentes, a correção de processamento é invocada. Caso eles estejam presentes ou após a correção, permissões de acesso e extensão de segmento são verificadas e o endereço físico

é calculado (simultaneamente). Se não ocorreram erros, R e M são atualizados. Quando o processo se completa, o endereço físico encontra-se na saída.

Quando um descritor de segmento ou de página necessário para uma tradução não está presente nos caches, a MMU acessa as tabelas de descritores em memória para carregá-los apropriadamente. Essa atividade é chamada correção de processamento. Ela pode ser necessária para carregar um descritor de segmento ou de página, ou ambos. Erros podem ocorrer durante essa atividade. Descritores inválidos (V==0) ou não presentes (P==0) nunca são carregados nos caches.

Os bits R e M dos descritores de página são sempre mantidos atualizados tanto no cache quanto na memória. O processo de verificação e sinalização de R e M é chamado atualização de R e M. Esse processo pode invocar correção de processamento para carregar um descritor de segmento (porque o descritor de segmento contém o endereço do descritor de página em memória).

3.6.4.2 ERROS

Se o processo de tradução de endereço não puder ser completado por algum motivo, um erro será acionado. Todos os erros fazem com que a MMU sinalize apropriadamente os campos FLTCR, copie o conteúdo do VAR no FLTAR e sinalize um erro para a CPU. Os conteúdos de FLTCR e FLTAR não serão alterados no curso de traduções bem-sucedidas (sem erro) e correção de processamento. Apenas outro erro ou uma escrita em modo periférico podem alterá-los.

O FLTCR contém três campos: requisição de acesso, acesso ao nível X e tipo de erro. O campo de requisição de acesso contém um código que especifica o tipo de acesso (leitura de instrução, leitura de dados etc.) tentado pela CPU quando da ocorrência do erro. O campo de acesso ao nível X contém um código que especifica o nível de execução no qual o erro de acesso ocorreu.

O campo de tipo de erro do registrador de erros de código (FLTCR) pode conter diversos valores correspondendo a diferentes tipos de erro. Eles se enquadram em diversas categorias do ponto de vista do sistema operacional. Enganos do usuário podem causar os erros de deslocamento no segmento, acesso, acesso e deslocamento no segmento, extensão da SDT, extensão da PDT ou SD inválido. Uma necessidade de troca pode ser sinalizada pelos erros de segmento não-presente, PDT não-presente ou página não-presente. Problemas sérios com o sistema operacional causam a sinalização de página dupla, correção de processamento na memória, atualização de R e M na memória ou muitos erros de acessos

indiretos. Os outros tipos de erro são objeto de *trap*, escrita de página e nenhum erro.

3.6.4.3 MODO PERIFÉRICO MAPEADO EM MEMÓRIA E DESCARREGAMENTO

A MMU pode ser usada como um periférico mapeado em memória. Desse modo muitos dos registradores internos da MMU e seus caches de descritores podem ser endereçados como posições de memória. Cada uma dessas posições pode ser lida ou escrita.

Um modo de acesso tipo periférico ocorre para a MMU quando a lógica externa sinaliza o pino de seleção da MMU. Diversos bits de endereço enviados à MMU são usados para selecionar um objeto dentro da mesma e (para alguns objetos) uma palavra dentro do objeto.

Todos os objetos descritos na Seção 3.6.3 podem ser acessados em modo periférico.

Escritas no FLTCR causam o campo de tipo de erro a ser sinalizado como nenhum erro.

Quando a CPU escreve um endereço na seção RAM A (SRAMA), a MMU descarrega todos os descritores SDC e PDC que se encontram na seção correspondente à recentemente escrita entrada da SRAMA. Quando a CPU escreve um endereço no registrador de endereço virtual (VAR), a MMU descarrega quaisquer entradas de cache de descritores de segmento ou página correspondentes ao endereço virtual.

3.7 CONSIDERAÇÕES DE SISTEMA OPERACIONAL

É responsabilidade do sistema operacional iniciar a MMU e todo o sistema sinalizando SDT e PDT em memória física e escrevendo valores nos registradores da MMU que provêm informações necessárias, tais como tabelas de endereços.

Os sistema operacional também é responsável por alterar valores de SRAM quando do chaveamento de processos. Isso fará com que a MMU descarregue e preencha seus caches.

A transferência de descritores entre os caches da MMU e memória física (correção de processamento) é tratada por ela sem a interferência do sistema operacional.

A ação do sistema operacional é requerida quando a MMU sinaliza para a CPU o

erro que ocorreu. Há um número de tipos de erro que se relacionam a erros que a MMU detecta em dados críticos, tais como erros de memória quando a MMU tenta ler SDT ou PDT. Esses erros requerem ações não-usuais e talvez drásticas por parte do sistema operacional.

Dois outros tipos de erros são página não-presente (para um segmento paginado) e segmento não-presente (para um segmento contíguo). Em ambos os casos a MMU está avisando à CPU que uma página ou segmento requerido não está presente em memória física, e, portanto, deve ser carregado. O sistema operacional é responsável por essas ativicades e deve efetuar qualquer E/S necessária para ajustar as entradas apropriadas na SDT e/ou PDT.

A MMU provê suporte de hardware para algoritmos de troca de página ou segmento por parte do sistema operacional, sinalizando os bits R (referenciado) e M (modificado) nos descritores de segmento e página sempre que já modificação ou referência numa página ou segmento. Se o sistema operacional limpa periodicamente os bits R, por exemplo, ele pode usá-los para implementar uma variação do algoritmo de troca LRU. Ele poderia escolher trocar segmentos ou páginas que permanecem com seus bits R limpos quando a troca é chamada, assumindo que esses segmentos ou páginas têm sido menos referenciados ultimamente que aqueles que com bits R sinalizados.

O sistema operacional irá ocasionalmente alterar os conteúdos das tabelas de descritores em memória. Por exemplo, ele precisa fazer isso para sinalizar e limpar bits P (presente) sempre que páginas ou segmentos são trocados em memória física. Qualquer alteração nos conteúdos das tabelas tem de ser seguida por algum tipo de descarregamento dos caches da MMU para prevenir a confusão que resultaria do fato de caches e tabelas conterem informações conflitantes.

3.8 ARQUITETURA DA UNIDADE DE ACELERAÇÃO MATEMÁTICA (MAU) WE32106

Essa unidade (MAU)[4] provê o microprocessador WE32100 com um suporte do padrão IEEE completo para ponto flutuante como um co-processador. A MAU suporta operações tanto em modo co-processador como em modo periférico. Dados de ponto flutuante de precisão simples (32 bits), dupla (64 bits) ou estendida (80 bits) têm suporte. Além disso, os inteiros de 32 bits e um tipo de dado decimal de 18 dígitos são suportados para conversões. Todos os quatro modos IEEE para arredondamento são suportados (para o mais próximo, para zero, para mais infinito e para menos infinito), bem como os cinco tipos de exceções mascaráveis (operação inválida, overflow, underflow, divisão por zero e

inexato). Além disso, uma exceção mascarável de overflow de inteiro é suportada para conversões de inteiros e decimais. A MAU suporta as seguintes operações:

1. *Aritméticas.* Soma, subtração, multiplicação, divisão, resto, raiz quadrada, negação, valor absoluto, comparação (quatro versões), movimentação.

2. *Conversões.* Ponto flutuante para inteiro, inteiro para ponto flutuante, ponto flutuante para decimal, decimal para ponto flutuante.

3. *Controle e Estado.* Leitura de registradores de estado, escrita nos registradores de estado, extração de resultados em erros, carregamento de registradores de dados, nenhuma operação.

As instruções de controle e estado permitem que o contexto da MAU seja salvo e restaurado em chaveamento de processos e também provêem informações de grande valor para depuração quando uma operação gera uma exceção.

A MAU suporta todos os tipos de dados requeridos, incluindo zeros positivos e negativos, infinitos positivos e negativos e detecção e não detecção de NAN (não-numérico). Underflow gradual é também suportado na forma de números normais em precisão simples e dupla e números não-normais em precisão estendida. Indicadores de sinal, zero e overflow são automaticamente retornados para a CPU, e o uso das instruções da MAU com a CPU WE32100 é totalmente transparente para o usuário em modo co-processador. Além disso, a MAU é compatível com todas as características internas do WE32100, incluindo DMA, interrupções e capacidade de reiniciação.

A MAU contém muitas otimizações de hardware para atingir alto desempenho em ponto flutuante. Os caminhos de interconexação de dados foram cuidadosamente otimizados para obter um alto nível de desempenho nas operações de soma e subtração. A adição de uma chave de barreira completa (BSW) é de especial ajuda nessas operações. A operação de multiplicação é implementada com uma forma altamente codificada do algoritmo de Booth, que processa três bits do resultado por iteração em uma malha fortemente canalizada. A operação de divisão usa uma implementação fortemente canalizada do algoritmo de divisão sem restauração. Além dessas otimizações internas, a interface de co-processador da MAU permite transferências excepcionalmente rápidas de dados para a CPU e memória num barramento de 32 bits. O efeito combinado dessas otimizações é uma taxa total de execução para o teste de Whetstone em precisão simples de 1,0 Mega Whetstone. Esse número é para a CPU WE32100 e MAUWE32106 rodando juntas em modo co-processador com zero estados de espera no acesso à memória a 14 MHz. Taxas de execução em dupla precisão são ligeiramente inferiores.

3.8.1 SUPORTE AOS PADRÕES IEEE PARA PONTO FLUTUANTE NA MAU WE32106

O padrão IEEE para aritmética binária de ponto flutuante facilita:

- Geração de software numérico de alta qualidade.
- Portabilidade de software entre implementações.
- Tratamento cuidadoso e determinístico de condições de exceçao e anomalias.

O padrão não impõe quaisquer restrições de implementações. Isso significa que uma implementação pode ser realizada totalmente em software, totalmente em hardware ou em qualquer combinação dos dois. A WE32106 é uma implementação em hardware do padrão.

3.8.1.1 FORMATOS DE DADOS

Um número em ponto flutuante consiste em três partes: um expoente assinalado, um significando (ou mantissa) e um único bit na posição mais à esquerda, que indica o sinal do significando. Este consiste em bits implícitos ou explícitos à esquerda do ponto binário implícito e fração à direita. O padrão divide os dados em básico e estendidos, com cada um possuindo duas precisões, simples e dupla. Toda a implementação deve suportar o formato simples. O suporte ao formato estendido é fortemente recomendado.

A MAU suporta formatos simples, duplos e duplos estendidos do padrão do IEEE.

3.8.1.2 TIPOS DE DADOS

A MAU suporta todos os tipos de dados requeridos. O que se segue é uma breve discussão desses tipos de dados.

Números normalizados em ponto flutuante. Esse é o tipo de dado de trabalho com o qual o usuário irá interagir. É definido pelo valor do expoente, não estando nos formatos máximo e mínimo.

Números não normalizados em ponto flutuante. Esses números possuem uma mantissa não-nula com o expoente correspondendo ao formato de valor mínimo. O bit implícito mais significativo da mantissa (à esquerda do ponto binário) possui valor zero. Um

número não normalizado é gerado quando uma exceção de underflow mascarada ocorre. Números não normalizados provêem underflow gradual para zero.

Zeros: este é um número com o expoente no formato mínimo e um significando zero. Zeros são assinalados.

Infinitos: um número com significando zero e formato de máximo expoente é infinito. Mais um vez há codificões para infinitos positivos e negativos.

Não-numérico (NaN): NaN são entidades simbólicas providas para propósitos de diagnóstico e para permitir melhorias que estão fora do escopo do padrão. NaN são números que possuem formato de máximo expoente com uma fração não nula. NaN são classificados em duas categorias – *sinalizante* e *quieto*. Um NaN quieto pode ser usado para detectar um erro que se propagou através de uma seqüência de operações. Pode também ser usado para indicar o uso de variáveis não iniciadas. Um NaN propaga-se por meio de operações sem causar exceções. Quando um NaN sinalizante é encontrado, uma exceção ocorre. Se a condição de exceção está mascarada, uma operação com um NaN sinalizante como operando completa-se normalmente e produz um NaN quieto como resultado.

3.8.1.3 OPERAÇÕES

As implementações segundo o padrão devem prover operações de soma, subtração, multiplicaçao, divisão, raiz quadrada, resto, arredondamento para valor integral, conversões entre vários formatos de inteiros em ponto flutuante para formatos decimais e comparação. A MAU suporta todos esses tipos de operação diretamente em hardware.

3.8.1.4 ARREDONDAMENTO

O arredondamento toma um número infinitamente preciso e o modifica para caber no formato de destino. O padrão requer que todas as operações em ponto flutuante produzam um resultado intermediário que está correto com precisão infinitas, sem limites de faixa, e então arredonde-o de acordo com o modo de arredondamento selecionado. O padrão propõe quatro modos diferentes de arredondamento:

- Para o mais próximo. Nesse modo, o valor representável mais próximo ao resultado infinitamente preciso é entregue. Esse é o modo default de arredondamento.
- Para infinito positivo.
- Para infinito negativo.

– Para zero.

A MAU implementa diretamente todos os quatro modos.

3.8.1.5 EXCEÇÕES

O padrão IEEE especifica cinco tipos de exceções: operação inválida, underflow, overflow, divisão por zero e inexato. O padrão requer controle de habilitação/desabilitação de trap individual para cada uma das exceções. A resposta default é proceder sem o trap. Flags individuais de estado são também requeridos para cada exceção. Os indicadores de estado podem ser permanentes, isto é, uma vez indicada a exceção será necessária uma operação explícita de limpeza do mesmo. A condição permanente dos indicadores permite que a informação de exceção se propague numa série de operações quando o trap correspondente está desabilitado. O tratamento de exceção na MAU é conforme o padrão descrito acima.

3.8.2 SUMÁRIO DO PADRÃO IEEE

O padrão IEEE para aritmética binária de ponto flutuante difere da aritmética genérica de processadores de ponto flutuante em diversos aspectos. Algumas características do padrão que aumentam a complexidade de hardware são:

1. O padrão recomenda três tipos de dados: simples, duplo e estendido. O número de bits é diferente em cada formato. Isso faz com que a conversão de um formato em outro seja mais complexa que outras implementações de ponto flutuante, nas quais precisões simples e duplas possuem expoentes de formato idêntico.

2. Os tipos de dados especiais tais como não-numérico (NaN), infinito, números desnormalizados e não normalizados requerem hardware especial para reconhecimento e processamento.

3. O padrão define cinco exceções com controles individuais de habilitação/desabilitação e indicadores de estado correspondentes. O padrão é também bastante específico sobre a resposta default à exceção quando está mascarada e a respeito das informações fornecidas à rotina de tratamento quando a mesma é habilitada.

4. A definição da operação de resto requer que ela seja quebrada em diversos passos, de forma a evitar interrupções que prenderiam o processamento.

3.9 ARQUITETURA DO CONTROLADOR DE ACESSO DIRETO À MEMÓRIA – DMAC

O DMAC WE32104 é um controlador de quatro canais projetado para aplicações em 32 bits. Uma arquitetura única de duplo barramento permite interconexão simples e eficiente de periféricos orientados a byte, tais como UART, controladores de disco e interfaces de rede, com um barramento de processador de 32 bits. Isolando o tráfego periférico mais lento num barramento separado, a lógica de interconexão fica não somente simplificada como a contenção pelo barramento do sistema fica extremamente diminuída, aumentando, portanto, a capacidade geral do sistema.

Para dar suporte a sua arquitetura de duplo barramento, o DMAC provê um buffer de dados de 32 bits para cada canal. Dados lidos do barramento periférico de largura de byte são aglutinados em operandos mais largos (até palavras quádruplas de 16 bytes), antes de serem escritos na memória. Além de transferências de byte, meia palavra e palavra, o DMAC suporta transferências de palavras duplas e quádruplas no barramento do sistema, visando melhorias na largura de banda da memória e redução do overhead de arbitração de barramento. Um diagrama em bloco do DMAC é mostrado na Figura 3.15.

Cada um dos quatro canais é controlado independentemente e pode ser programado nos modos de transferência em *burst* ou por roubo de ciclo. No modo *burst*, o DMAC irá segurar o barramento do sistema por tanto tempo quanto necessário, enquanto no modo de roubo de ciclo o barramento é liberado após cada transferência. Há quatro tipos de transferências: memória para periférico, periférico para memória, memória para memória e preenchimento de memória. Outras opções programáveis incluem seleção de características de transferência do dispositivo periférico, habilitação de interrupções ao final da transferência e habilitação de um canal para transferência encadeada. No modo encadeado, uma lista ligada de blocos de requisição contendo fonte/destino e contagem de transferência é carregada em memória pelo processador. O DMAC pode então executar essa lista, carregando parâmetros de transferência diretamente da memória sem qualquer outra intervenção do processador central.

No modo *burst*, um DMAC de 14 MHz pode executar cópias de memória para memória a 11,2 Mbyte/s e operações de preenchimento de memória a 20,3 Mbyte/s. Dado que essas taxas são cinco vezes maiores que aquelas que podem ser conseguidas através de instruções da CPU, muitas funções do sistema podem ser codificadas para alto desempenho usando o DMAC.

Para transferência para/do barramento periférico, a máxima taxa de transferência é 6 Mbyte/s; no caso de *burst* é 3,7 Mbyte/s se apenas um byte for transferido a cada

requisição do periférico. É igualmente importante que o DMAC somente requisite o barramento do sistema quando uma palavra quádrupla foi aglutinada em seu buffer de dados, executando em um ciclo de barramento o que de outra forma necessitaria de dezesseis ciclos. Incluindo o overhead durante a arbitração de barramento, a utilização do mesmo é reduzida por um fator de dez.

3.9.1 CÓPIA MEMÓRIA – A – MEMÓRIA PELO DMAC MELHORA PERFORMANCE DO UNIX

As operações de cópia memória-a-memória e preenchimento de memória pelo DMAC são até cinco vezes mais rápidas que as operações equivalentes executadas pela CPU. Rotinas lógicas do sistema UNIX, tais como fork, exec e E/S lógico entre áreas do sistema e do usuário, baseiam-se em cópias rápidas de buffers e limpeza de blocos, que podem ser otimizadas usando-se um canal do DMAC. Levantamentos com o sistema UNIX demonstram que a CPU despende aproximadamente 10% de seu tempo executando essas rotinas. Usando-se as capacidades do DMAC pode-se, portanto, melhorar o desempenho do sistema em 8%.

3.9.2 VANTAGENS DO BARRAMENTO PERIFÉRICO DE 8 BITS NO DMAC

O barramento periférico de 8 bits é idealmente adequado a periféricos de 8 bits, como controladores de disco e interfaces de rede local. A separação do barramento periférico do barramento de sistema de 32 bits reduz a contenção pelo mesmo. Isto permite à CPU a execução mais eficiente enquanto o DMAC está servindo periféricos. Num modelo, a CPU possuía dez vezes a alocação do barramento do sistema comparado a outro sem o barramento periférico, quando uma transferência de periférico de 10 Mbit/s estava sendo servida pelo DMAC.

3.10 ARQUITETURA DO CONTROLADOR DE RAM DINÂMICA WE32103

O controlador de RAM dinâmica (DRC) provê multiplexagem de endereço, gerenciamento de acesso e tempo de ciclo e controle de refrescamento para memória dinâmica de acesso randômico (DRAM). A extrema flexibilidade do DRC permite uma ampla seleção de componentes DRAM a serem usados em diversas configurações de sistema.

Entre as características suportadas pelo WE32103 estão: diversos modos de refrescamento (distribuído/*burst*, temporizado internamente/externamente), constantes de tempo programáveis para casar-se com os requerimentos dos diversos componentes

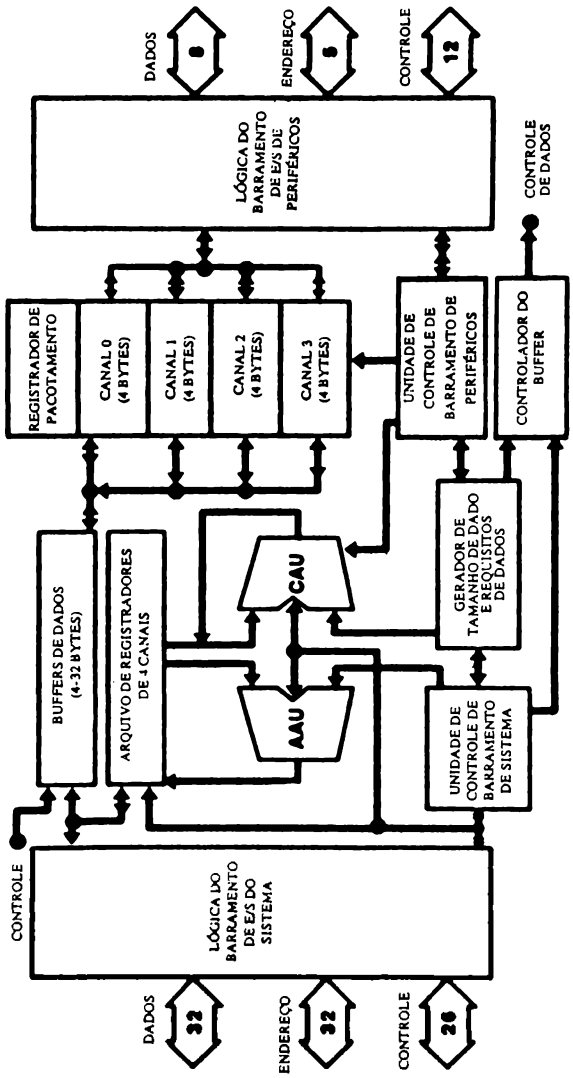


Figura 3.15 Diagrama de blocos do controlador de acesso direto à memória WE32104.

DRAM e suporte a muitas extensões de DRAM, tais como 256 k por 1,64 k por 4 e 1 M por 1. O DRC é capaz de endereçar DRAM de até 16 Mbytes usando pastilhas de 1 Mbyte. Modos de operação em página e nibble podem ser selecionados para acessos rápidos de palavras duplas ou quádruplas. Além disso, um modo especial de pré–tradução é provido para melhorar o tempo de acesso em sistemas incorporando memória virtual paginada. Para aplicações que requerem operações altamente confiáveis, um conjunto completo de sinais de controle é provido para detecção de erro (opcional) e correção por hardware.

O DRC possui um modo assíncrono que permite que ele opere com outros microprocessadores comerciais. No entanto, um melhor desempenho pode ser atingido no modo síncrono, quando configurado com o conjunto WE32100. A operação síncrona irá, em geral, ganhar um estado de espera em operações de leitura e escrita do processador. O modo de pré-tradução reduz ainda mais o tempo de acesso, deixando de lado a linha de memória que está sendo usada num acesso da MMU WE32101. Finalmente, ciclos rápidos de palavras duplas e quádruplas, utilizando DRAMs em modo paginado ou nibble, são suportados para melhorar o desempenho tanto da CPU quanto do DMAC.

3.10.1 MODO DE PRÉ-TRADUÇÃO DO WE32103 USANDO DRAMS DE 256 K

O DRC possui a capacidade de iniciar um acesso de memória antes que a MMU tenha traduzido o endereço virtual. Esse início adiantado é vantajoso em sistemas utilizando um esquema de memória virtual paginada. Nesses sistemas, os bits menos significativos do endereço são o deslocamento de página, e, portanto, equivalentes em espaços virtuais e físicos.

Se programado para um RAS adiantado, o DRC irá utilizar o deslocamento de página dentro do endereço virtual como endereço de linha e gerar o habilitador de endereço de coluna (RAS0) em paralelo com a tradução de endereço da MMU. Utilizando a característica de pré-tradução, um ciclo pode ser ganho no acesso de memória. Um diagrama em bloco do DRC é mostrado na Figura 3.16.

3.11 SUMÁRIO

O conjunto WE32100 e seu predecessor (WE32000) são operacionais em muitos sistemas AT&T, em aplicações de compartilhamento de tempo, tempo real e orientadas à transação.

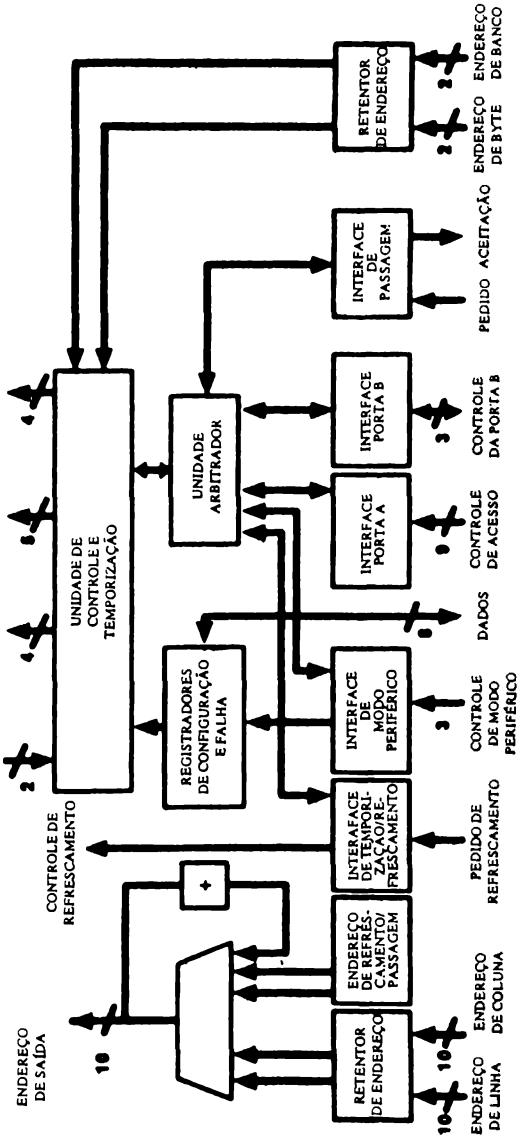


Figura 3.16 Diagrama de blocos do controlador de RAM dinâmica WE32103

O conjunto WE32000 é usado nos sistemas 3B2 e 3B5. A CPU é também usada como um controlador de periféricos inteligente para o sistema 3B5. Outras aplicações incluem o terminal Teletype DMD 5420, onde o WE32000 é usado para executar primitivas gráficas, bem como programas residentes. Interfaces de rede usando o WE32000 estão disponíveis para tratar controle de fluxo e protocolos de alto nível nos sistemas 3B. Placas baseadas em barramento VME foram construídas para tomar vantagem das capacidades do conjunto de pastilhas e podem ser usadas como blocos de construção de computadores de placa única ou sistemas de múltiplos processadores. Diversos sistemas a multiprocessadores também foram construídos.

O conjunto de pastilhas é apoiado por um conjunto completo de ferramentas de desenvolvimento baseadas em UNIX que provêem acesso fácil a sinais internos e estado para depuração e desenvolvimento de hardware, e depuração simbólica para desenvolvimento de software[5].

Com o alto desempenho da MAU, o conjunto de pastilhas possui um desempenho bastante competitivo em operações de ponto flutuante e abriu as portas para as mais amplas aplicações (por exemplo, CAD/CAM), que requerem aritmética de ponto flutuante e alta precisão. O microssistema UNIX está se expandindo para desempenho ainda mais alto, caminhando para menores regras de projeto e implementações melhoradas, enquanto mantém compatibilidade com os produtos disponíveis hoje. A adiantada disponibilidade do DMAC WE32104 e do DRC WE32103 enfatizam a preocupação da AT&T com uma solução completa. A MMU WE32101 provê capacidades de gerenciamento de memória não igualadas por nenhum produto correntemente no mercado.

AGRADECIMENTOS

Este capítulo foi escrito em conjunto por muitas pessoas que trabalharam na arquitetura e projeto das pastilhas. Algumas seções foram extraídas de publicações anteriores[2,3]. Gostaria de agradecer as contribuições escritas das seguintes pessoas: A. D. Berenbaum, M. W. Condry, W. A. Dietrich, J. A. Fields, M. L. Fuccio, A. K. Goksel, L. N. Goyal, U. V. Gumaste, M. E. Thierbach e P. A. Voldstad.

REFERÊNCIAS BIBLIOGRÁFICAS

1. "Hardware Configuration and I/O Protocol of the WE32100 Microprocessor Chip Set". Fuccio, M. L. e Goyal, L. N., *WESCON 1985*.

2. "The Operating System and Language Support Features of the BELLMAC-32 Microprocessor". Berenbaum, A. D., Condry, M. e Lu, P. M. *Symp. Arch. Support for Prog. Lang. and Op. Sys.* Palo Alto, Cal., 30-38, 1-3, mar., 1982.

3. "Architecture of a VLSI Map for BELLMAC-32 Microprocessor". Lu, P. M., Dietrich, Jr., W. A., Fuccio, M. L., Goyal, L. N., Chen, C. J. Blahut, D. E., Fields, J. A., Goksel, A. K. e LaRocca, F. D. *Spring COMPCON 1983,* São Francisco, Ca., 213-217, 28 fev. a 3 mar., 1983.

4. "An IEEE Standard Floating Point Chip". Goksel, A. K., Diodato, Phil W., Fields, John A., Gumaste, Ulhas V., Kung, Chew K., Lin, Kingyao, Lega, Mario E., Maurer, Peter M., Ng, Thomas K., Oh, Yaw T. e Thierbach, Mark E. *ISSCC 1985*, 18 e 19, fev. 1985.

5. "Hardware/Software Development System for the WE32100 CPU and MMU". Clark, M. H., Rango, R. A., Rusnock, K. J. e Stubblebine, W. A. *WESCON 1984.*

CAPÍTULO **4**

O TRANSPUTER DA INMOS

J. R. Newport
CAP Scientific Ltd

4.1 INTRODUÇÃO

Os transputers, fabricados na Grã-Bretanha pela Inmos, são computadores de única pastilha de alto desempenho com enlaces (links) seriais completos processador-processador e uma memória em pastilha opcional. Entretanto, tal descrição simples subestima o potencial dessa classe de dispositivos.

O nome *transputers* deriva de duas palavras: *transistor* e *computador*. A intenção é justamente a de que como os transistores têm sido blocos para construção de computadores, então transputers serão os blocos para a construção de toda uma nova classe de máquinas, desde compactos processadores vetoriais até supercomputadores.

Tentativas para construir-se sistemas de multiprocessadores a partir de microprocessadores convencionais sempre têm sido repletas de problemas que não são menores que aqueles concernentes à comunicação interprocessos. Normalmente são encontrados problemas de largura de banda quando mais de quatro processadores são ligados por um barramento compartilhado. Esse efeito pode ser claramente percebido quando examinamos o gráfico do desempenho em relação ao número de processadores (veja Figura 4.1 (*a*)). O desempenho variará dependendo dos tipos de processadores e barramentos usados, mas se atingirá um ponto em que, somando-se processadores, o barramento será estrangulado, diminuindo então o desempenho total do sistema. Esse limite geralmente é alcançado quando seis a oito processadores compartilham o mesmo barramento.

Os transputers têm seus próprios enlaces seriais, cada um com uma capacidade de entrada e saída concorrente de 10 Mbit/s. Um transputer típico tem quatro enlaces desse tipo, proporcionando uma capacidade de comunicação de 40 Mbit/s para recepção e

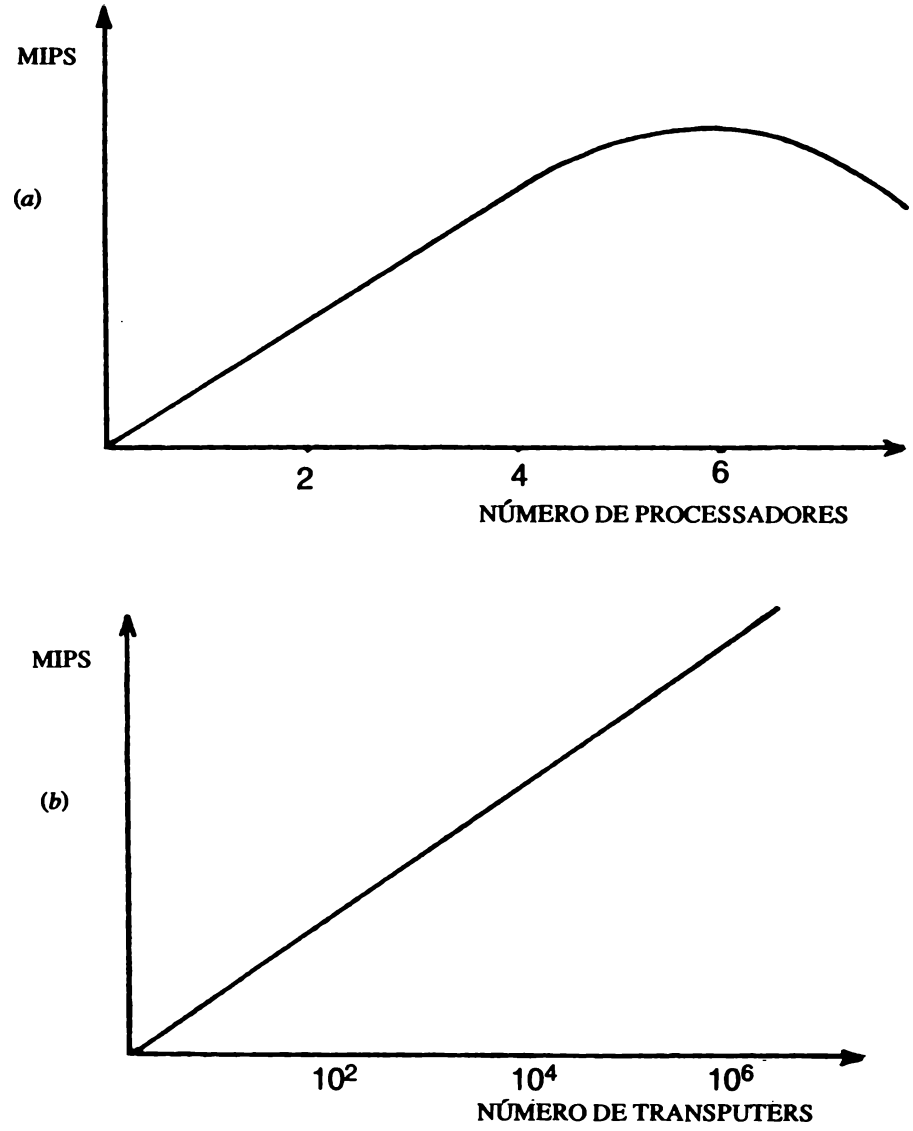


Figura 4.1 (*a*) Diagrama de desempenho *versus* número de processadores para um sistema multiprocessador convencional; (*b*) o mesmo diagrama para um sistema multitransputer.

40 Mbit/s para transmissão. Como esses enlaces são uma parte integral de cada transputer individual, quanto maior o número de transputers em uma rede, maior a largura de banda total do sistema. Devido à banda de comunicação ser um dos problemas, o gráfico do desempenho do sistema em relação ao número de processadores para uma rede de transputers mostra um resultado aproximadamente linear (veja Figura 4.1 (*b*)).

O limite em que transputers conectados rigidamente em rede, por seus enlaces seriais, se torna impraticável, não foi ainda determinado. Em teoria, não há razão para que supercomputadores contendo milhares ou mesmo centenas de milhares de transputers não possam ser construídos.

Uma área onde dispositivos de alto desempenho são necessários é a de Inteligência Artificial (IA). O desempenho de máquinas de IA é medido em deduções lógicas por segundo (Logical Inferences Per Second – LIPS), com uma dedução lógica sendo equivalente a uma centena de instruções de máquina. O objetivo da equipe japonesa no desenvolvimento de quinta geração de arquiteturas foi uma máquina protótipo capaz de um bilhão (ou gita) de deduções lógicas por segundo para 1991. Acredita-se que tal máquina poderia, por exemplo, ser capaz de traduzir japonês para inglês (e vice-versa) em tempo real, usando tanto texto como voz na entrada. Um vetor de 10 mil transputers poderia potencialmente alcançar um desempenho similar, ser montado e estar funcionando em breve. O desenvolvimento de software de uma aplicação particular pode bem ultrapassar essa data por um ou dois anos, dependendo da complexidade da aplicação. Entretanto, o hardware para uma máquina de quinta geração poderia estar disponível em cinco anos acima do que foi considerado ser um objetivo ambicioso por muitos.

Como a Inmos percebeu que simplesmente fornecendo um hardware simples com alto potencial de desempenho não se produz necessariamente um sistema de alto desempenho; uma nova linguagem de programação, occam, tem sido desenvolvida juntamente com o transputer. Linguagens melhor instituídas como Pascal, FORTRAN e C serão também disponíveis para uso nos transputers, mas é o occam que habilitará que o potencial de desempenho dos sistemas baseados em transputers seja completamente obtido. Occam provê diretamente suporte de software para concorrência e para enlaces seriais interprocessadores de 10 Mbits/s.

A arquitetura do transputer combina várias características desenvolvidas para outros processadores através dos anos para alcançar o desempenho estimado de 10 milhões de instruções por segundo (10 MIPS). Entretanto, os elementos-chave que permitem aos sistemas de alto desempenho serem construídos a partir de dispositivos transputer são concorrência e comunicação e o suporte provido para ambas as características pela linguagem occam.

4.2 A ARQUITETURA GENÉRICA DO TRANSPUTER

Como se pode ver na Figura 4.2, a arquitetura do transputer não define um simples dispositivo, mas, ao contrário, toda uma família de transputers. O transputer de 32

bits T424, com quatro enlaces seriais e 4 kbytes de memória RAM estática em pastilha, é usualmente tido como um transputer padrão de propósito geral, mas várias combinações de facilidades em pastilha são possíveis. Os transputers sempre incluem um processador, serviços de sistema e um ou mais enlaces seriais, mas eles podem ter 4 kbytes de RAM em pastilha, 2 kbytes, ou não ter memória em pastilha.

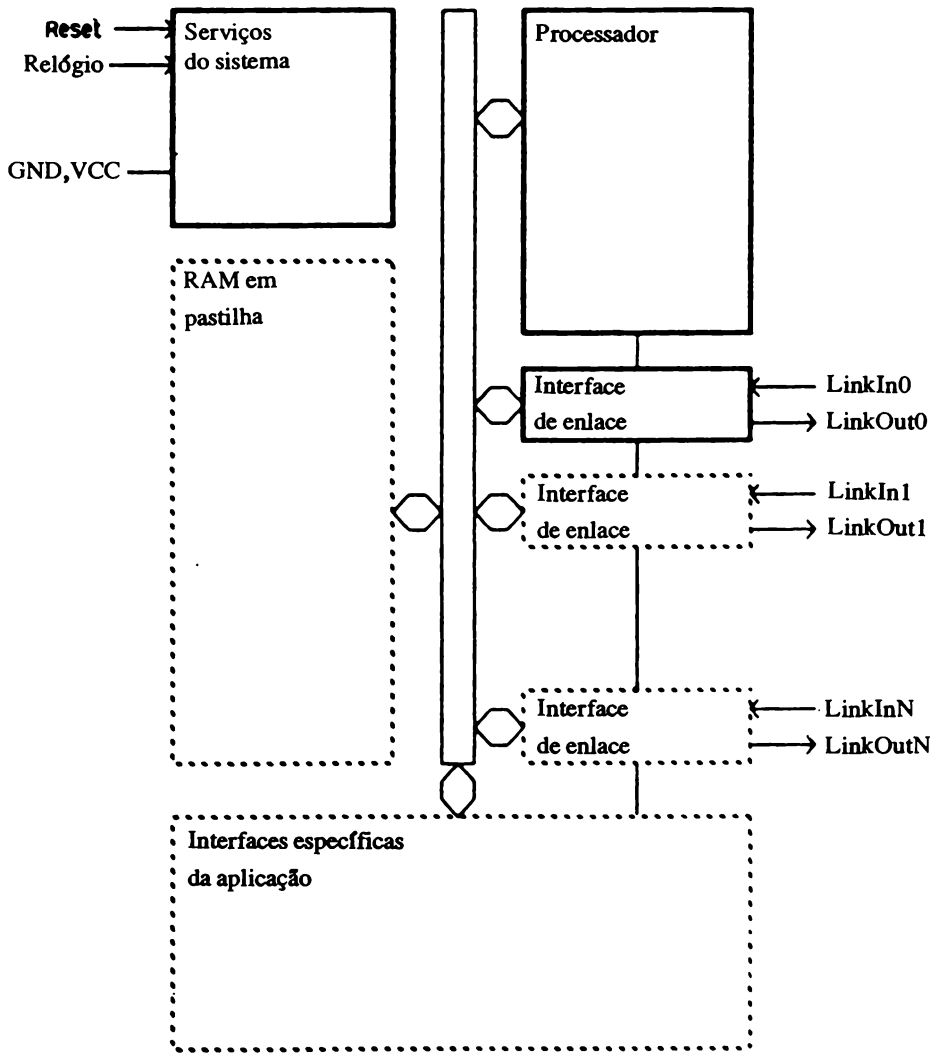


Figura 4.2 Arquitetura geral do Transputer (*Cortesia do grupo de companhias Inmos*).

O primeiro transputer de 32 bits a tornar-se disponível foi o T414, liberado pela Inmos em outubro de 1985. O T414 tem 4 enlaces seriais, mas somente 2 kbytes de RAM em pastilha. Uma família de transputers de 16 bits também está sendo desenvolvida, sendo o primeiro destes o T212. Liberado ao mesmo tempo que o T414, o T212 também tem 2 kbytes de memória em pastilha.

O segundo dígito no nome do produto transputer indica o tamanho da memória RAM em pastilha em unidades de 2 kbytes. Um 4 seguindo o T indica um processador de 32 bits, e um 2 na mesma posição, um processador de 16 bits. Entretanto, a Inmos tem sido menos lógica com o último dígito, o qual afirma prover uma singularidade do produto em vez de indicar o número de enlaces disponíveis em pastilha (o T212 tem 4 enlaces, não 2, como poderia ser esperado). A velocidade de um dispositivo particular em megahertz (MHz) pode também ser incluída no nome, seguida por hífen; por exemplo, um T424-20 poderia rodar a 20 MHz, e um T424-12 poderia rodar a 12,5 MHz (a velocidade dos transputers aumenta a passos de 2,5 MHz, de 10 até 20 MHz).

Todos os transputers têm um temporizador que corre um relógio em separado do relógio externo de 5 MHz. Ambos, a freqüência de relógio do processador e o relógio do enlace (o qual controla a taxa de transferência dos enlaces seriais), são derivados da freqüência de relógio externa, por escalonamento interno. Transputers futuros serão capazes de operar seus enlaces em velocidades maiores, mas todos suportarão a velocidade de 10 Mbit/s como padrão. Os pinos de seleção de velocidade em cada dispositivo permitem que um ou mais enlaces sejam selecionados para a velocidade-padrão, permitindo comunicação entre dois transputers quaisquer. Uma tolerância de mais ou menos 0,02% é permitida no sinal de relógio externo para operação correta dos enlaces seriais. Isto significa que dois transputers comunicando-se não têm de usar o mesmo relógio externo. A tolerância também é tal que cristais osciladores relativamente baratos sejam usados para gerar os sinais. Os enlaces são previstos para serem usados somente entre transputers na mesma placa, ou entre placas adjacentes no mesmo bastidor. À longas distâncias (>400 mm) os enlaces precisarão de alguma forma de compensação (buffering). Os drivers/receptores RS422 podem ser usados para estender a distância de comunicação de enlace.

O protocolo de enlace, como a velocidade de enlace, é padronizado através de toda a classe de transputers. Os dados são transmitidos um byte por vez em pacote, com cada pacote sendo confirmado pelo transputer receptor. O pacote de confirmação é enviado logo que o pacote de dado é identificado, e, desde que cada enlace seja bidirecional, pacotes de dados e pacotes de confirmação podem ser enviados concorrentemente. Pacotes de confirmação consistem em 2 bits, um 1 e um 0. O segundo bit identifica o tipo de pacote. Os pacotes de dados consistem em dois 1s seguidos pelo byte de dado e finalizado com 0 (veja Figura 4.3).

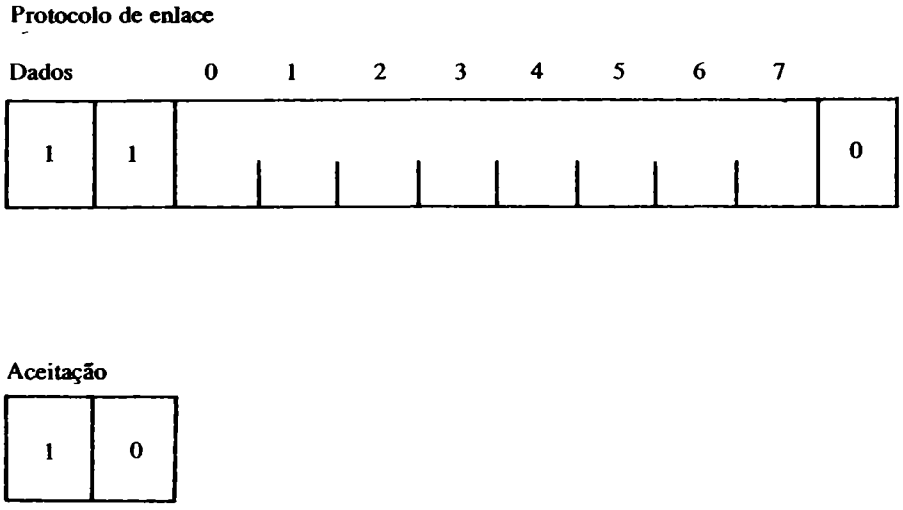


Figura 4.3 Protocolo de enlace (*Cortesia do grupo de companhias Inmos*).

O T2i2 e T414 são degraus no caminho do desenvolvimento do T222 e T424, respectivamente. As características dos produtos transputers no futuro serão governadas mais e mais pelas necessidades específicas do usuário. Na medida em que experiência é adquirida em sistemas de desenvolvimento e em muitos casos no desenvolvimento de produtos incorporando os processadores T414 e T424, essas necessidades específicas tornar-se-ão mais evidentes. Algumas aplicações podem ser beneficiadas por terem mais RAM em pastilha, enquanto outras podem requerer mais enlaces. Como os enlaces e a memória em pastilha disputam espaço na pastilha, haverá sempre um compromisso entre os dois, com o balanço ideal dependendo da aplicação. Em um sistema de processador simples, por exemplo, pode ser mais útil ter apenas um ou dois enlaces para entrada e saída e ter mais RAM em pastilha, que poderá ser acessada mais rapidamente do que memória RAM externa. Isto poderia melhorar todo o desempenho de processamento.

Muitas das aplicações previstas do T424 têm processadores arranjados em grandes vetores de duas dimensões, com cada T424 conectado a seus quatro vizinhos mais próximos via seus enlaces seriais (veja Figura 4.4). Logo que os projetistas de sistemas se familiarizem com a construção de redes de transputers, eles começarão a pensar: "Por que conectar apenas os quatro vizinhos mais próximos, por que não oito?" Engenheiros mais aventurosos já estão planejando redes tridimensionais, requerendo no mínimo 6 enlaces e até redes de dimensão n requerendo/2n enlaces em cada transputer. O problema em construir-se redes de transputers tridimensionais pode ser resolvido, em termos simples, emparelhando-se os transputers. Conectando-se 2 T424, usando um enlace cada, liberam-se 6

enlaces, disponíveis para conectar os seis vizinhos mais próximos em três dimensões. A necessidade de hipercubos, superclusters e outras ousadas arquiteturas serão consideradas mais detalhadamente na Seção 4.10.

As características de transputers futuros dependerão da demanda, investimento e avanços da tecnologia. Não seria realista esperar um transputer de 6 enlaces com 8 kbytes de memória RAM em pastilha (um T446, se uma nomenclatura mais lógica fosse adotada) ser lançado imediatamente depois do T424. Mas se a demanda existe, uma pastilha com 6 enlaces, com 2 kbytes, pode ser inteiramente praticável (T416), possivelmente seguida por uma versão com 4 kbytes de memória em pastilha (T426).

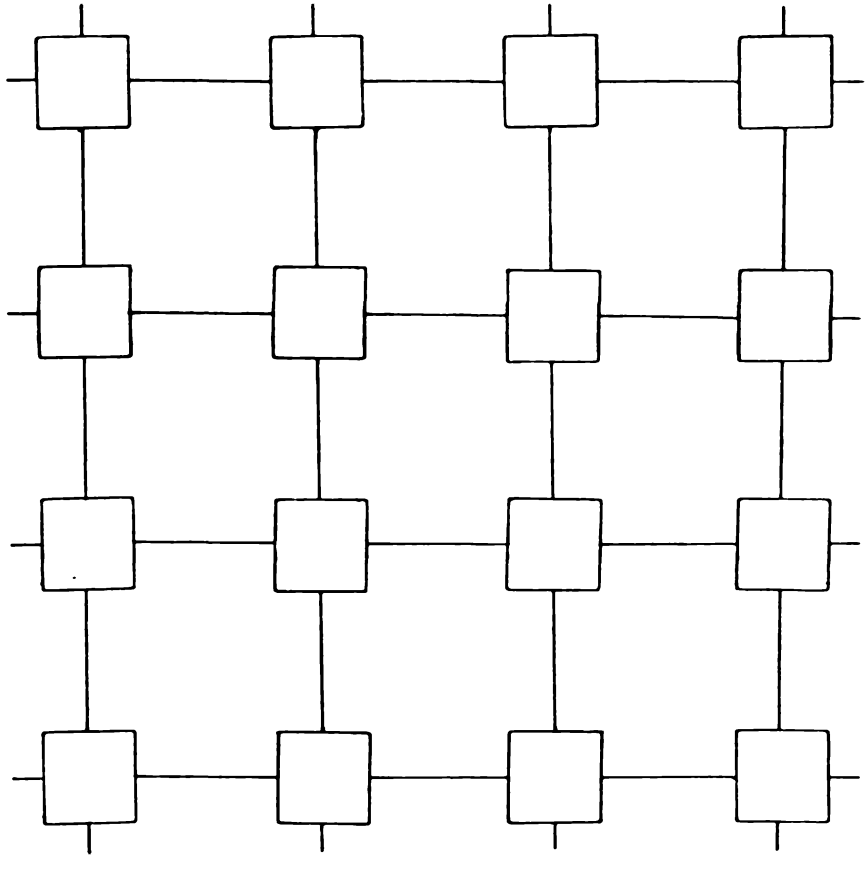


Figura 4.4 Vetor bidimensional de transputers.

Em cada adição a enlaces seriais e memória em pastilha, a arquitetura genérica de transputer também provê a inclusão de interfaces para aplicações específicas em pastilha. Transputers de propósitos específicos para gráficos e controladores de acionadores de disco já foram desenvolvidos e liberados em 1985. Estes são o controlador gráfico G412 e o controlador de disco M212 (o M provê de Mass storage). Esses transputers especializados provêem um meio direto de comunicação, conveniente com os dispositivos periféricos. Embora esses não sejam transputers reais, é válido mencionar os adaptadores de enlace Inmos nesta seção. A série C de dispositivos, C001, C002 etc., são dispositivos especiais que podem ser ligados a um enlace serial do transputer para adaptá-los a enlaces paralelos de 8 bits. Um terceiro método para interconectar periféricos é mapeá-los em memória no barramento do transputer. Um periférico pode ser mapeado em qualquer endereço da memória externa e pode ser controlado usando-se instruções escritas em occam.

O transputer para ponto flutuante, o F424, está em desenvolvimento e pode ser liberado alguns meses depois do T424. O F424 não será um co-processador para o T424, como tem sido a prática para microcomputadores convencionais, como a combinação Intel 8086/8087, mas será um transputer completo por si só. Ele reterá todas as funcionalidades do T424 em adição às facilidades aritméticas de ponto flutuante, conforme o rascunho-padrão 10,0 IEEE P754, implementado em pastilha. Operações em precisão simples (bit de sinal, expoente de 8 bits e fração de 32 bits) e precisão dupla (bit de sinal, expoente de 11 bits e fração de 52 bits) terão suporte para números com ponto flutuante. É prevista pela Inmos que operações com números de ponto flutuante levem aproximadamente 1 microssegundo, dando ao F424 uma avaliação de pico de desempenho de um milhão de operações de ponto flutuante por segundo (1 MFLOPS).

Muitas aplicações, tais como simulação, processamento de sinal e a análise de dados sísmicos, requerem que números grandes de operações de ponto flutuante sejam realizadas no menor tempo possível. Tal como os projetistas de computadores de quinta geração têm um alvo de 1000 LIPS, também aqueles envolvidos com o desenvolvimento de vetores de processadores e mainframes com aceleradores de ponto flutuante têm um alvo de 1000 MFLOPS. Uma máquina construída a partir de mil transputers F424 poderia potencialmente fornecer tal desempenho – um giga FLOPS.

4.3 O TRANSPUTER T424

Tendo-se considerado as características gerais da família de transputers, um membro específico da família, o T424, será agora considerado (veja Figura 4.5). O T424 tem 4 kbytes de memória RAM estática em pastilha com um tempo de acesso de 50 ns, 4 enlaces seriais bidirecionais de 10 Mbits uma larga interface de 32 bits para 25 Mbytes de

memória externa e naturalmente um processador e serviços de sistema. Isto representa aproximadamente 250.000 dispositivos fabricados usando um processo CMOS de 1,5 μm. A pastilha T414, o primeiro transputer de 32 bits, ocupa uma área de silício de mais de 9 × 9 mm e contém 200.000 componentes. Poderia esperar-se que o T424 fosse um pouco maior, embora os projetistas objetivem minimizar qualquer acréscimo no tamanho devido às restrições de embalagem. É assumido que o T424, como o T414, será disponível em uma pastilha padrão de 84 pinos J-lead carrier.

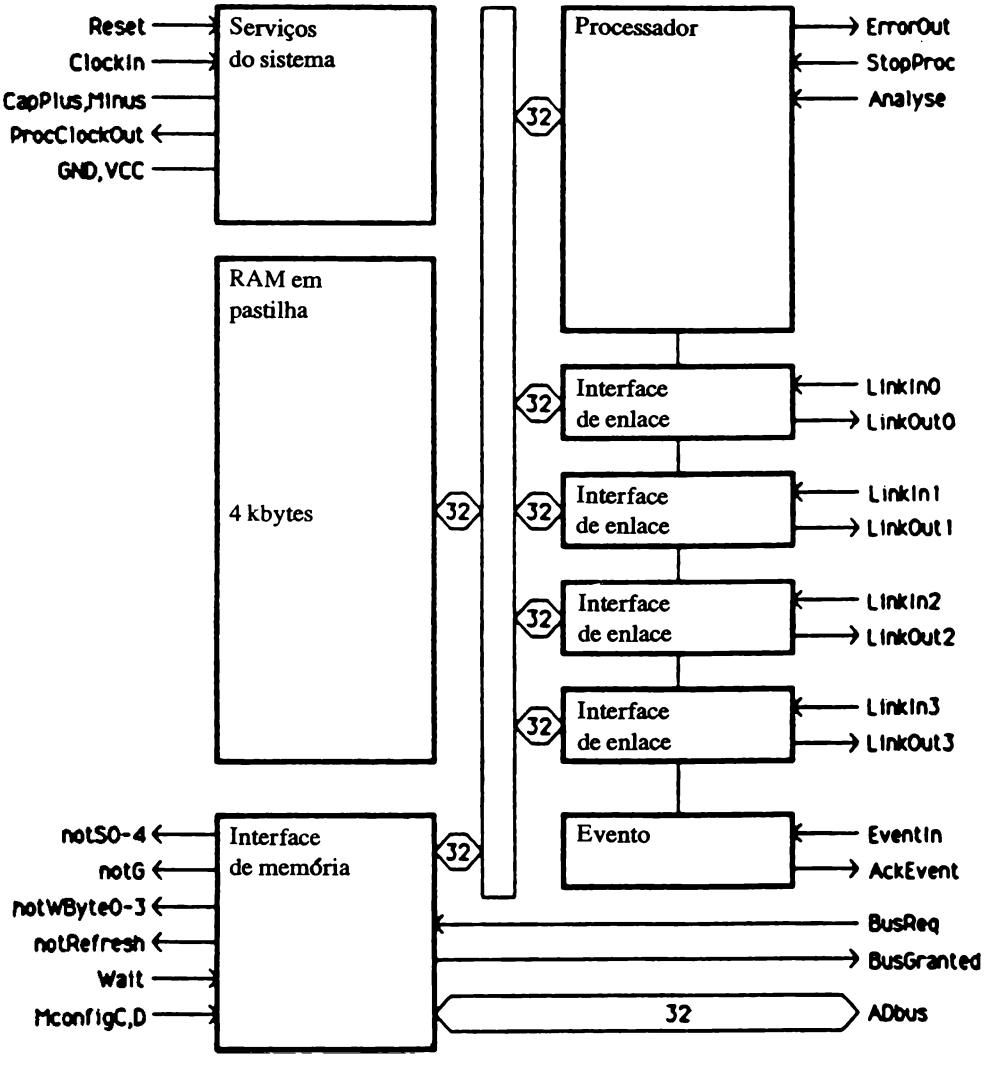


Figura 4.5 O T424 (*Cortesia do grupo de companhias Inmos*).

Como a maioria dos processadores de 32 bits, o T424 é habilitado para endereçar diretamente 4 Gbytes (2 elevado à potência de 32) de memória. Os endereços de memória são sinalizados, o que significa que eles podem ser manipulados como qualquer outro inteiro de 32 bits. A Figura 4.6 mostra o mapa do espaço de memória do T424. O código básico em ROM (Ready Only Memory) é, por convenção, alocado no fim (mais positivo) do espaço do endereçamento, e quando configurado para bootstrap a partir da ROM, o processador irá começar a execução a partir do endereço hexa 7FFFFFFE. A RAM em pastilha é organizada em 1 k palavras de 32 bits cada, situadas no final mais negativo da memória, isto é, no endereço hexa 80000FFF a 80000000. Oitenta palavras (72 bytes) de memória em pastilha, do endereço hexa 80000047 a 80000000, são usadas pelo sistema, as

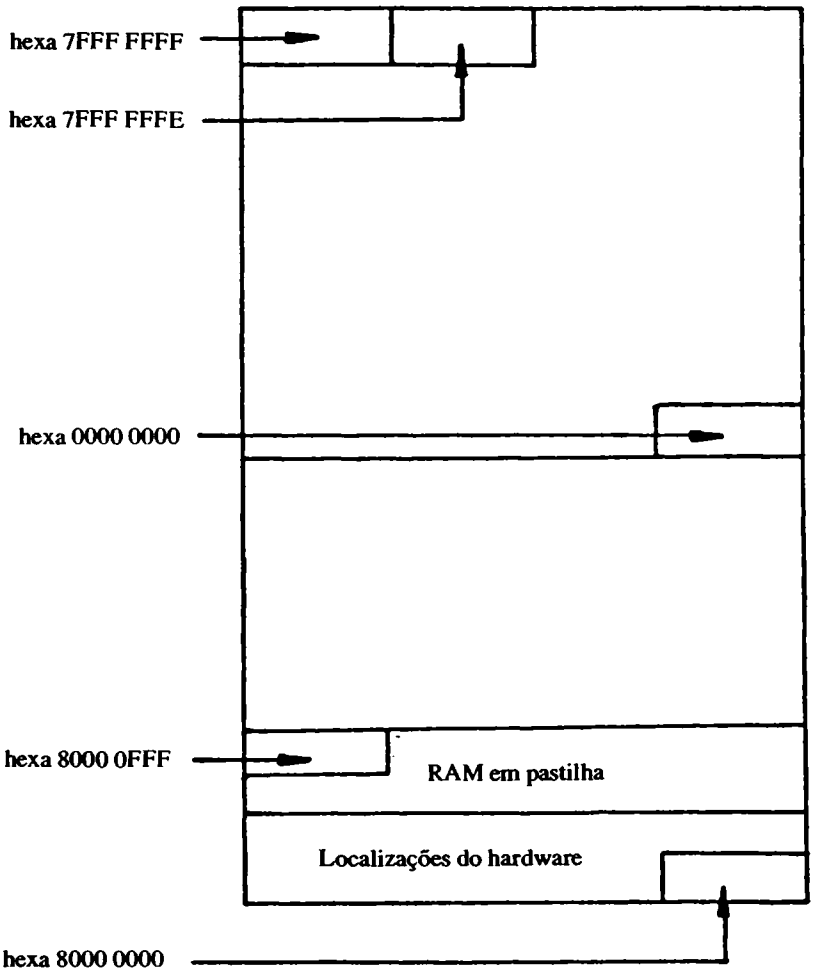


Figura 4.6 Mapa de memória do T424 (*Cortesia do grupo de companhias Inmos*)

últimas oito palavras nessa ordem são associadas aos quatro enlaces (uma palavra de entrada e uma de saída para cada enlace).

O restante da memória RAM em pastilha (80000FFF a 80000048) é destinado para código de aplicações e dados. Desde que a memória em pastilha pode ser acessada em 5 ns, o que é tipicamente a metade do tempo para acesso à RAM externa, programas pequenos que requerem alto desempenho podem ser idealmente alocados exclusivamente em RAM em pastilha. Onde são usados vetores de transputers interconectados por seus enlaces seriais, pode ser possível subdividir programas maiores e alocá-los apenas em memória RAM. Se isso não for possível, os processos mais críticos em tempo precisam ser identificados para que possam ser alocados na memória em pastilha. A linguagem occam provê facilidades para alocar código em endereços específicos no espaço de memória do transputer. Essas facilidades serão explanadas futuramente na Seção 4.6.

Alguns microprocessadores fazem uso de memórias caches de alta velocidade que podem estar dentro ou fora da pastilha. Assim como o processador executa uma instrução de um programa, um microcódigo dedicado busca (prefetch) um bloco de código que inclui a instrução imediatamente seguinte à corrente apontada pelo ponteiro de instrução e aloca esse código no cache. Quando o ponteiro de instrução chega ao fim do cache ou quando uma instrução de Jump (salto) aponta para uma instrução fora do código contido no cache, o microcódigo irá automaticamente buscar um novo bloco de código e o alocará no cache. Uma vantagem de memória cache é que o usuário não tem de estar ciente que ele existe. Por outro lado, RAM em pastilha em transputers é tratada pelo processador do mesmo modo que uma RAM externa – ele simplesmente leva menos tempo de acesso. Isso significa que o ônus de se fazer o melhor uso da rápida RAM fica para o usuário. Se isto é considerado uma vantagem ou desvantagem dependerá da necessidade e preferência do usuário.

É útil lembrar-se de que uma instrução de transputer tem o tamanho de apenas um byte, mas a memória é acessada sempre por palavras. Portanto um acesso à memória externa buscará ou quatro instruções ou um dado de 32 bits. A Inmos conduziu um experimento para demonstrar os efeitos de alocação de ambos: dados e códigos em pastilha. Os tempos dados na tabela seguinte são relativos ao caso onde todo o programa e todos os dados são alocados em pastilha, sendo normalizados.

Altamente orientado a:	*Dados*	*Programa*
Todo o programa e dados na RAM em pastilha	1	1
Programa fora, dados em pastilha	1,3	1,1
Programa em pastilha, dados fora	1,5	1,2
Programa e dados fora da pastilha	1,8	1,3

Esses dados são ferramentas muito importantes, particularmente para aplicações que têm um código grande mas poucos dados (menos que 4 kbytes). O processador T424 também incorpora um buscador (pre-fetcher) de instrução, o qual busca 8 instruções (duas palavras). Isto tem sido visto como um compromisso ideal entre a redução de acesso ao código e busca de instruções que podem não ser executadas.

A tendência geral dos microprocessadores tem sido a inclusão de mais e mais registradores, mas o T424 usa apenas seis para executar um programa seqüencial. Esses consistem em um registrador de operando, um ponteiro de instrução, um ponteiro de área de trabalho e três registradores que fazem avaliação da pilha. Em adição a esses seis, há mais dois registradores que são usados pelo escalonador de prioridade da pastilha para compartilhar o tempo entre processos paralelos alocados no mesmo transputer. Embora occam permita qualquer número de níveis de prioridade, somente dois são suportados pelo T424 e T414, sendo estas a prioridade 0 e prioridade 1, com a 0 sendo a mais alta. Se um ou mais processos de prioridade 0 forem alocados em um transputer, um deles será selecionado pelo escalonador e executado completamente, ou até que tenha de esperar por comunicação ou por entrada do relógio. Os processos de prioridade mais alta devem ser mantidos relativamente pequenos, do contrário eles tenderão a monopolizar o processador. Os processos de prioridade 1 serão escalonados somente quando os processos de prioridade 0 não puderem seguir. Os processos de prioridade 1 executam em partes de tempo, de modo que nenhum processo tenha de esperar muito tempo. As partes de tempo nas versões anteriores de transputer usavam 4096 ciclos dos 5 MHz de relógio de entrada. Em dispositivos posteriores, entretanto, foi estendido para 5120 ciclos do relógio de entrada (input clock), o que corresponde a um tempo de mais de 1 ms.

4.4 O CONJUNTO DE INSTRUÇÕES DO T242

Os transputers são uma forma de reduzir o conjunto de instruções de um computador (RISC). As características e vantagens de usar um RISC são bem documentadas e foram vistas no Capítulo 2. Em resumo, foi visto que programas que rodam em computadores com conjunto de instruções complexo (CISC) usam somente uma pequena porcentagem dessas instruções em uma base regular. Algumas das instruções restantes são usadas somente em raras situações. Os projetistas de máquinas RISC concentraram seus esforços em dar suporte às instruções mais freqüentemente usadas, com as demais sendo implementadas usando-se combinações de instruções ou adicionando-se modificadores ou extensões. Algumas dessas funções podem requerer mais ciclos de máquina para executar em RISC que em CISC. Entretanto, como um todo, os RISC alcançam um alto desempenho e eficiência devido à requisição de menos esforços para decodificação de instruções e requerendo menos registradores.

Falando-se genericamente, projetar um compilador para um RISC é uma tarefa relativamente imediata. O compilador é apresentado como uma tarefa relativamente simples e prognosticada, e o código objeto tende a ser compacto e eficiente. Embora uma tarefa relativamente simples para uma máquina, gerar código para um RISC tende a ser tedioso e inclinado a erros quando feito por um programador humano. Isto pode bem ter sido um fator contribuinte para a decisão da Inmos de não abandonar detalhes como valores de código de instrução ou mesmo toda a gama de instruções oferecidas em um transputer particular. O occam é pretendido como a linguagem de mais baixo nível usada em qualquer transputer, e a Inmos não suporta o uso de código de máquina pelo usuário. Há de fato mais razões em volta dessa aparente decisão do que a preocupação com inclinação a erros em programação. A tecnologia VLSI está avançando rapidamente e estão sendo desenvolvidas técnicas onde novas características podem rapidamente ser incluídas no projeto de hardware da pastilha. Mantendo-se o occam como linguagem padrão para todos os membros na família de transputers, um novo dispositivo pode ser desenvolvido com um conjunto de instruções totalmente novo e com, por exemplo, um aumento significante de desempenho. Tudo que o usuário teria de fazer seria comprar um novo compilador para o dispositivo, e ele poderia rodar todos os seus programas occam antigos. Mudanças menores em um programa podem ser necessárias para otimizar o desempenho quando transportando entre transputers. Isto seria particularmente verdade se um novo dispositivo incorporasse mais RAM em pastilha ou tivesse mais enlaces seriais. Ter um compilador separado para cada dispositivo é mais problemático que manter versões separadas dos mesmos programas no que seria efetivamente linguagens diferentes. A informação seguinte é incluída puramente como matéria de interesse, já que é improvável que o leitor tenha de programar em código de máquina de transputer. Nenhum valor é dado para as várias funções e instruções desde que estes podem variar, mesmo entre versões diferentes do mesmo dispositivo.

Todas as instruções no transputer são de um byte dividido em dois campos. Os quatro bits mais significativos contêm a função e os quatro menos significativos contêm o dado (veja Figura 4.7). Campos de quatro bits permitem dezesseis itens de dados locais serem manipulados por dezesseis instruções. Trinta desses valores são usados para representar as funções usadas mais freqüentemente. Estas são conhecidas como *instruções de um endereço,* já que o operando é usado pela instrução como um valor. Essas instruções são:

Load local
Store local
Load local pointer
Load non-local
Store non-local

Load non-local pointer
Adjust workspace
Load constant
Add constant
Add to memory
Jump
Conditional jump
Call

Duas instruções adicionais, Prefix e Negative Prefix, podem ser usadas para estender o operando de qualquer instrução para qualquer tamanho. Todas as instruções começam carregando os quatro bits menos significativos do registrador de operando com os quatro bits de dado do campo de instrução, como mostrado na Figura 4.7. Todas as instruções, exceto a Prefix e a Negative Prefix, terminam limpando o registrador de operando para a próxima instrução. A instrução Prefix carrega seus quatro bits de dado no registrador de operando e então desloca-os de quatro para a esquerda (isto é, os quatro bits mais significantes). A instrução de Negative Prefix é similar, exceto que ela complementa o registrador de operando antes de deslocar. Usando-se uma instrução de prefixo simples, qualquer operando de –256 a 255 pode ser representado. Usando-se várias instruções de prefixos, qualquer operando de até 32 bits pode ser representado.

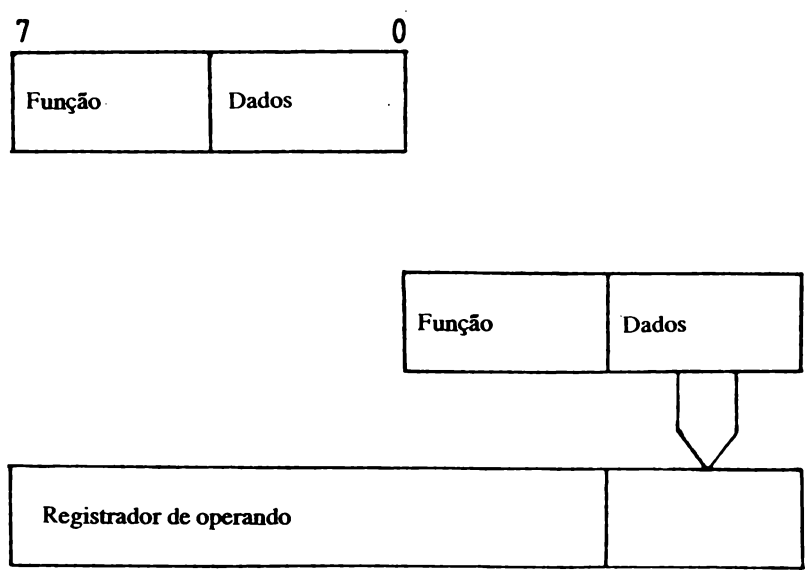


Figura 4.7 Formato de instrução do T424 (*Cortesia do grupo de companhias Inmos*).

Os outros tipos de instrução são as de endereço zero. Estas usam o operando para definir operações com valores já armazenados na pilha de avaliação:

Add	Shift left	Read timer
Subtract	Shift right	Test error
Multiply		Reverse
Divide	Move message	Return
Remainder	Input message	Minimum integer
Long Add	Output message	Initialise
Long subtract		
Long multiply	Load byte	Start process
Long divbide	Store byte	End process
Normalise	Word count	Alt start
	Byte subscript	Enable channel
Difference	Word subscript	Disable channel
Greater than	Check subscript	Disable timer
Equal zero	Extend to word	Alt wait
	Check partword	
And	Extend to double	
Or		

4.5 HISTÓRIA E FILOSOFIA DA LINGUAGEM OCCAM

O nome occam provém de William of Occam, um filósofo inglês do século quarto que estudou em Oxford. William possui o crédito de uma citação em latim conhecida como Occam's Razor:

Entia non sunt multiplicanda praeter necessitatem

No qual os estados das entidades não devem ser multiplicados sem necessidade. No segundo encontro do grupo de usuários de occam, ocorrido apropriadamente na cidade natal de William of Occam, Oxford (home town of Oxford), um orador apontou o fato de ele ter dúvidas sobre um filósofo que conjecturou tal forma complexa de dizer:

MANTENHA AS COISAS SIMPLES

Essa tradução menos formal do razor de Occam apresenta muito mais claramente uma visão da filosofia em torno da linguagem. A simplicidade do occam é um reflexo da simplicidade do subjacente conjunto de instruções. A linguagem occam e o transputer foram desenvolvidos de mãos dadas para produzir uma linguagem que combinasse os beneff-

cios de desempenho da linguagem assembly com legibilidade, produtividade do programador e aspectos de manutenção de uma linguagem de alto nível.

Em termos de complexidade, o occam pode ser visto como sendo o outro extremo do espectro do Ada, o novo padrão de linguagem da US DoD para sistemas. Os projetistas do Ada encontraram-se em uma tarefa extremamente difícil na qual lhes foi dada uma lista extensa de características essenciais que a nova linguagem padrão teria de suportar. Não somente a linguagem Ada teria de suportar características de programação como construções de estruturas e diversos tipos de dados, também foi requisitado suporte para princípios de engenharia de software, tais como abstração, segurança de informação e modularidade. É discutível se esses aspectos de manipulação de software de alto nível poderiam ser alcançados (addressed) pelas características da linguagem por si mesma. Algumas sugestões do tipo "como um sistema grande em occam poderia ser desenvolvido e manejado?" serão dadas em uma seção posterior.

Uma indicação da complexidade de uma linguagem de programação é a medida do tamanho da definição da linguagem. Como um teste para julgar as outras duas linguagens, Algol 60 pode ser definido em 43 páginas de texto, com 50 linhas de texto em cada página, e Pascal pode ser definido em 38 páginas. Occam, por sua vez, em 18 páginas. Quando ser breve não é levado em consideração, não será surpresa descobrir-se que ADA requisitou uma definição de 275 páginas. Embora isto não seja de forma alguma uma medida da complexidade da linguagem, uma comparação útil é fornecida. Ao menos uma idéia do tamanho do problema que o escritor do compilador de Ada enfrenta.

À parte terem sido nomeados através de uma figura histórica em vez de uma abreviação, as linguagens occam e Ada têm uma característica importante em comum: ambas suportam processos concorrentes e também, em cada uma, o modelo de concorrência foi baseado no trabalho conduzido pelo professor Hoare, da Universidade de Oxford, sobre processos seqüenciais comunicantes. O professor Hoare estava envolvido com o estágio de definição da linguagem occam, que foi desenvolvida na Inmos por Davis May.

Concorrência tem sido reconhecida como uma característica-chave para liberar o projetista de sistema de limitações físicas do hardware disponível. O desempenho na execução seqüencial de software é limitado pela velocidade de um processador simples na qual ele roda. Somente dividindo-se o software em tarefas que possam ser executadas independentemente, e em paralelo em processadores separados, o desempenho poderá ser aumentado. Rodar um software totalmente independente em paralelo nunca apresentou problemas. Uma companhia com uma força de trabalho em expansão, dividida igualmente em duas fábricas, que não tenha como rodar os programas de folha de pagamento de ambas em seu computador, simplesmente compra um outro computador. Sistemas de tempo

real para aplicações críticas apresentam problemas muito maiores. Suponhamos que um processador particular tenha sido escolhido para rodar um sistema de controle de tráfego aéreo. É requisitado que o sistema detecte mais de cem aeronaves ao mesmo tempo. Devido a funções extras e checagens de segurança especificadas no sistema, os desempenhos calculados indicam que um dos processadores escolhidos pode detectar apenas uma aeronave por vez. Poderia uma aeronave ser traçada em cada processador e suas saídas serem combinadas de alguma forma? Obviamente não. O processo de software modelando cada aeronave precisa ser capaz de examinar os estados dos processos modelando outra aeronave. Isto pode ser feito tanto diretamente como via controle de processos que planejam movimentos futuros da aeronave e prevêem quando uma aeronave pode aproximar-se muito de outra. Características essenciais que poderiam ser suportadas por uma linguagem de tempo real poderiam ser concorrência e comunicação.

Ada provê códigos objetos conhecidos como tasks (tarefas) para suportar concorrência, mas são apenas um dos tipos de código no qual os programas Ada podem ser construídos. No occam, o objeto que suporta concorrência é o process (processo), que é de fato o único tipo de objeto dentro da linguagem occam. Os programas occam são construídos exclusivamente em níveis hierárquicos de processos. No nível mais baixo, a linguagem é construída através de apenas três tipos de primitivas de processos: *assignment, input* e *output* (associação, saída e entrada). As primitivas de saída e entrada suportam diretamente comunicação entre os processos chamados via canais.

No início desta seção, foi colocado que a filosofia na linguagem occam era a de se manter as coisas simples. Podemos ver agora que isto pode ser alcançado reconhecendo-se que uma linguagem de tempo real precisa suportar apenas duas características essenciais:

concorrência e comunicação.

4.6 LINGUAGEM OCCAM

Os programas occam são construídos em níveis hierárquicos de processos. No nível mais baixo, a linguagem é formada de apenas três tipos de primitivas de processos: *assignment, input* e *output* (atribuição, entrada e saída). O símbolo usado para atribuição é := como em várias outras linguagens de alto nível. Entretanto, no occam não é requerido nenhum símbolo para terminalizar uma primitiva de processo, mas apenas uma primitiva pode aparecer em cada linha. Por exemplo:

```
x := 1
```

atribui o valor da expressão à direita do símbolo := à variável previamente definida na esquerda; neste caso atribui o valor literal 1 à variável x.

O símbolo ? é usado para indicar input, por exemplo:

```
c ? x
```

lê um valor de um canal previamente definido, c, e atribui esse valor à variável x.

O símbolo ! é usado para indicar un output. Por exemplo:

```
x ! y
```

escreve o valor contido em y no canal c.

Podem ser estruturadas primitivas de processos de mais alto nível por meio de construções. A construção seqüencial, SEQ, indica que o processo que o segue deve ser executado seqüencialmente. Processos componentes são identificados por dois caracteres que respeitam a construção.

```
SEQ
      c1 ? x
      x := x + 1
      c2 ! x
```

No exemplo acima não há alternativa senão executar os três processos seqüencialmente. O valor deve ser lido de c1 e associado a x antes de ele ser incrementado, antes de ser escrito em c2. Entretanto se dois valores são lidos de dois canais diferentes em variáveis diferentes, então as duas leituras podem conceitualmente ser executadas em paralelo. Essa condição é fornecida pela linguagem occam e é implementada pela construção paralela, PAR.

```
PAR
      c1 ? x
      c2 ? y
```

Se x e y tiverem de ser somadas e o resultado associado a x, então obviamente isto não poderia ser feito até que as entradas tenham sido feitas. Similarmente, se as expressões contendo o novo valor de x devem ser escritas na saída, elas devem seguir o processo soma. O programa abaixo consiste em três processos seqüenciais. O primeiro consiste em um processo de duas entradas paralelas; o segundo sendo uma primitiva de processo (uma associação); e o terceiro consiste em um processo de duas saídas paralelas.

```
SEQ
      PAR
            c1 ? x
            c2 ? x
      x := x + y
      PAR
            c3 ! x + 1
            c4 ! x - 1
```

As posições dos três processos constituintes da construção seqüencial podem ser vistas como sendo alinhadas umas com as outras e recortadas em dois lugares com respeito a SEQ. Desta forma, mais e mais processos complexos podem ser construídos. Entretanto, a fim de incrementar a boa leitura e a manutenção de um programa, é de boa prática nomear os processos, ou PROC, cada um dos quais têm somente poucos níveis de processos constituintes. O processo acima poderia ser nomeado da seguinte forma:

```
PROC  P1 (CHAN c1, c2, c3, c4) =
      VAR x :
      VAR y :
      SEQ
            PAR
                  c1 ? x
                  c2 ? x
            x := x + y
            PAR
                  c3 ! x + 1
                  c4 ! x - 1
            SKIP :
```

O SKIP simplesmente termina sem efeito (note que o fim de uma PROC nomeada deve também ser indicado pelo símbolo :). Processos nomeados comunicam-se por meio de canais que conectam dois processos juntos. Quando ambos os processos, no fim de um canal particular, estão no mesmo transputer, a comunicação ocorre por meio de transferência de dados memória para memória. Se os processos estiverem alocados em transputers diferentes, a comunicação ocorre via um enlace padrão Inmos. Nos dois casos os processos serão concorrentes e a comunicação será sincronizada, com a transferência ocorrendo quando ambos os processos de entrada e saída estiverem prontos.

A alocação dos processos em um transputer específico é controlada por comandos de alocação. Ambos os processos compilados separadamente e componentes de uma

construção PAR podem ser alocados individualmente dessa forma. PLACE é também usado para associar canais nomeados conectando processos em transputers separados com os enlaces seriais reais.

PRI PAR aplica uma prioridade ao processo componente de uma construção PAR. Isto somente se aplica ao PAR mais exterior. A prioridade mais alta é dada ao primeiro do processo paralelo, prioridade 0, e ao segundo é dada a prioridade 1, e assim por diante. O número de níveis de prioridade depende da implementação do escalonador de prioridades em um transputer particular. O T414 e o T424 suportam dois níveis de prioridade, 0 e 1.

Pode-se usar repetições das construções SEQ para criar loops e PAR para construção de vetores de processos correntes.

```
SEQ i = 0 FOR n
  P
PAR i = 0 FOR n
  Pi
```

O primeiro exemplo executa o loop n vezes, o segundo irá criar n processos paralelos, P0 até Pn–1.

A construção IF permite um dentre um número de processos ser executado dependendo da condição que o precede:

```
IF
    x > 0
        x := x - 1
    x < 0
        x := x + 1
    x = 0
        SKIP
```

A construção ALT pode ser particularmente útil onde sinais podem aparecer em um dentre um número de canais alternativos:

```
ALT
      c1 ? ANY
           cont1 := cont1 + 1
      c2 ? ANY
           cont2 := cont2 + 1
      c3 ? ANY
           cont3 := cont3 + 1
```

O uso do ANY indica que o valor real dos sinais não é importante, somente a presença do sinal naquele canal particular é de interesse.

A repetição é implementada com o uso do WHILE:

```
WHILE (x - 1) > 0
    x := x - 1
```

A condição WHILE TRUE permite que os processos executem continuamente desde que iniciados:

```
PROC  quadrado (CHAN Xin, QuadradoOut) =
      WHILE TRUE
            VAR x:
            SEQ
                  Xin ? x
                  QuadradoOut ! x*x :
```

(Este exemplo também permite mostrar a sintaxe de declaração de parâmetros para canais, Xin e QuadradoOut.) Todas as implementações de occam suportam um número básico de tipos:

CHAN, TIMER, BOOL, BYTE e INT

Outras implementações também podem suportar os inteiros sinalizados INT16, INT32 e INT64; e números com ponto flutuante de acordo com o Padrão IEEE P754 versão 10.0: o REAL32 e o REAL64.

Podem ser construídas expressões usando-se os seguintes operadores:

Operadores aritméticos:	+, –, *, /, / (/ é resto)
Aritmética modular:	PLUS, MINUS, TIMES, DIVIDE
Relacional:	=, < >, >, <, > =, < =
Operadores Booleanos:	AND, OR, NOT
Operadores de bit:	BITAND, BITOR, > <, BITNOT (> < é OU exclusivo)
Operadores de deslocamento:	<<, >>

Todos os transputers incorporam um relógio que pode ser lido em uma variável do tipo inteiro:

```
tim ? t
```

Leituras com atraso também são suportadas e são da forma:

tim ? AFTER t0

4.7 DESENVOLVIMENTO DE SISTEMAS DE TRANSPUTERS

Em um ambiente ideal, o usuário provê ao projetista de sistemas um requerimento e uma especificação de desempenho, mas ele não especifica uma configuração de hardware particular. Deveria ser deixado que o projetista especificasse que hardware seria necessário para implementar seu projeto com o desempenho requisitado. A terminologia varia de aplicação para aplicação e de companhia para companhia, mas o desenvolvimento de software usualmente é dividido em três fases:

- especificação de solicitações;
- projetar;
- implementar.

Como ninguém é ingênuo para esperar que um sistema implementado funcione da primeira vez e continue a funcionar durante anos, é necessário considerar-se duas fases posteriores:

- teste e depuração;
- manutenção.

A fase inicial, especificação de solicitações, é simplesmente o processo de obter-se as requisições do usuário e apresentá-las numa forma que não seja ambígua e seja compreensível por ambos, usuário e projetista. Em um futuro não muito distante, máquinas mostrando algum grau de inteligência artificial serão capazes de executar diretamente a especificação de solicitações, mas até então é necessária uma fase de projeto para gerar um código objeto que possa ser compilado e rodado.

Experiências com o desenvolvimento de vários sistemas software grandes têm mostrado que, para produzir um código que tenha chances de estar correto e passível de suporte, é essencial o uso de um método de desenvolvimento. As metodologias de projeto mais promissoras aplicáveis ao projeto de software occam concorrente são aquelas baseadas em diagrama de fluxo de dados e software modular. Diagramas de fluxo de dados representam os projetos de hardware como uma rede local de processos conectados pelo curso de dados (veja Figura 4.8). Essa representação mapeia diretamente no modelo occam de conexão de processos por canais. Algumas metodologias de projeto, notadamente Jackson System Development (JSD Sistema de Desenvolvimento Jackson), usam proces-

sos de software para modular processos em ambientes reais. JSD considera esses processos uma rede local, cada qual considerado conceitualmente como sendo executado em seu próprio processador. Se há mais processos que processadores no hardware-alvo, então o processo é invertido. Inverter um processo é um meio de transformá-lo, e assim ele pode ser chamado como um procedimento. O escalonador do hardware do transputer pode rodar processos occam diretamente. Isto remove a necessidade da fase de inversão do JSD.

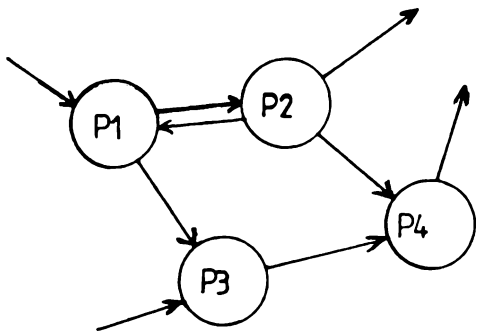


Figura 4.8 Processos occam vistos como um diagrama de fluxo de dados.

Independente do método adotado, um alto nível de automação será requisitado, particularmente se forem implementados projetos de software grandes em redes locais de transputers. Já existem algumas ferramentas de software, mas ainda são básicas e usadas isoladamente. Foi mencionado anteriormente que é discutível se uma linguagem por si forneceria facilidades para se verificar a exatidão do código, manipular-se os erros de tempo de execução e controle de configuração. Quando um software de conjunto de ferramentas completo e integrado tiver sido desenvolvido, não haverá necessidade de duplicá-las na linguagem de programação. O fato de occam prover apenas um número limitado de facilidades para gerenciamento do software e verificação de erros não será um problema então.

A área em que occam oferece várias vantagens positivas sobre outras linguagens é na sua flexibilidade na fase de implementação. Quando o projeto para um sistema tiver sido desenvolvido, ele pode ser implementado em occam em qualquer computador-mestre (host), para o qual um compilador tenha sido desenvolvido. Uma vez testado, esse mesmo código occam pode ser recompilado e rodado em um transputer simples ou em um vetor de transputers interconectados. Usando-se o operador PLACED, os processos podem ser alocados em um transputer específico. O desempenho estimado é obtido da Inmos, que roda em uma máquina-mestre (as séries DEC Vax 11/780 são suportadas atualmente como máquinas host) e permite que o desempenho no hardware-alvo seja previsto. Além disso, o número de transputers necessários para alcançar o desempenho especificado pode ser determinado antes de o software rodar no hardware-alvo. Se o desempenho requisitado é

aumentado pelo usuário, realocar o software em um vetor de transputer maior é relativamente uma tarefa simples.

Transputer e occam parecem oferecer uma combinação extremamente atrativa de hardware e software para desenvolvimento de sistemas concorrentes. Entretanto ainda há vários problemas para serem resolvidos. Para desenvolvimento de software para tempo real e geração de código sintaticamente correto não é o fim da história. Uma das características de software de tempo real é que o código testado no mestre pode criar um impasse (deadlock) quando rodado no hardware-alvo. O único caminho em que impasses (deadlock) podem ser permitidos é a provisão de mecanismos para proteger-se contra eles em todas as situações possíveis nas quais eles possam ocorrer. Os canais de interprocessos occam apresentam o maior potencial para ocorrência de impasses. Se por alguma razão um processo não for capaz de escrever em um canal, o processo no outro extremo do canal parará, o que poderia causar uma reação em cadeia e parar todo o sistema. Occam não fornece assistência para permitir ou recuperar-se de tal impasse. No terceiro encontro do grupo de usuários do occam (23-24 de setembro, 1985), o problema tido como o que necessitava de atenção mais urgente, foi o relacionado com os canais associados com impasses. Isto poderia não apresentar um problema maior para o desenvolvimento de sistemas baseados em transputers, mas é uma área que requer alguns esforços urgentes de pesquisa.

Para o futuro esperamos que em meses, em vez de em anos, um conjunto de ferramentas capaz de manejar um projeto de sistema moderadamente grande, implementado em occam tenha sido desenvolvido. Um programa occam que tenha sido testado em um hospedeiro (host) é alocado em um transputer simples e roda uma aplicação particular em três segundos. A especificação de desempenho, entretanto, estabelece uma requisição de um segundo. Vendo-se que o topo do nível de hierarquia consiste em três processos em uma construção paralela, o projetista muda o PAR para PLACED PAR, e aloca os três processos em três transputers separados emparelhados pelos seus enlaces seriais. A mesma aplicação está rodando, e o projetista é tomado pelo terror – dois segundos para rodar!

A resposta para esse problema é simples: um dos três processos paralelos no programa toma o dobro de tempo para executar que os outros dois somados. Esse processo, portanto, representa uma passagem crítica e precisa ser decomposto em processos menores. Esse exemplo trivial ilumina a necessidade de considerar o que pode ser chamado de primeira lei de processos concorrentes:

Divida carga de trabalho entre os processadores disponíveis.

A próxima seção irá considerar alguns dos outros problemas que precisam ser considerados quando se projeta e implementa software occam concorrente em redes de transputers.

4.8 COMPARTILHAR A CARGA

O segredo em tornar real todo o desempenho potencial de qualquer sistema de multiprocessadores está na distribuição da carga de trabalho. O paralelismo é o meio mais óbvio e mais atrativo de compartilhar a carga de trabalho entre os processadores, mas freqüentemente uma solução paralela para um problema não é simplesmente impossível. Considere o programa occam:

```
SEQ
    P1
    P2
    P3
```

Os processadores P1, P2 e P3 contêm totalmente seqüências diferentes de primitivas de process. P3 usa o resultado final de P2, e P2 usa o resultado final de P1; portanto a construção acima é a única forma de implementar o programa. Como pode a carga de tal programa seqüencial ser compartilhada entre multiprocessadores? A resposta é canalizar (pipelining) os processos do programa. Canalização envolve alocação seqüencial de processos em um arranjo linear de processadores.

Na Figura 4.9, cada um dos processos de software P1, P2 e P3 é alocado em seu próprio processador. P1 começa, lê um item de dado D1, processa esse dado e escreve em um resultado intermediário para P2 então termina; P2 começa, processa o resultado intermediário de P1, escreve seu próprio resultado intermediário para P3 e termina; P3 começa, roda e escreve o resultado final do programa R e termina.

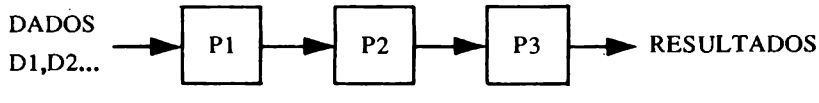


Figura 4.9 Transputers dispostos em canalização.

Será imediatamente aparente que somente um desses processos está executando por vez. De fato se todos os três processos pudessem ser alocados dentro da memória em pastilha de um transputer simples, ele poderia executar tão rapidamente em um processador quanto em três. As áreas onde canalizações são úteis são onde grupos de dados contínuos precisam ser processados. Considere um grupo de itens de dados D1, D2, D3, D4, ..., Dn sendo lidos na canalização acima. Dessa vez, quando P1 estiver processando D1, ele escreve o resultado intermediário para P2, mas não terminaliza. Ao invés, ele repete a seqüência com o item de dado D2. No segundo período de tempo, P2 está processando o derivado de D1 enquanto P1 está processando D2. Portanto ambos P1 e P2 estão operan-

do em paralelo. Se a carga de trabalho foi igualmente dividida entre os processadores, então cada conjunto de dados sobrará o mesmo tempo em cada processador, T. Depois do tempo 3T, D1 terá sido processado para produzir o resultado R1 no fim da canalização, e todos os processadores continuarão a executar enquanto um novo item de dado for lido em P1.

Pode-se ver agora que existem duas técnicas para compartilhar a carga de trabalho: canalização e paralelismo. Consideremos agora como as duas técnicas podem ser combinadas. Retornando ao problema anterior de como distribuir três processos não iguais capazes de processar em paralelo, como pode a carga de trabalho ser distribuída entre três processadores se P1 roda em quatro unidades de tempo e P2 e P3 rodam em uma unidade de tempo cada?

```
PAR
      P1
      P2
      P3
```

Rodando esses processos em um transputer simples são usadas seis unidades de tempo. Pode-se esperar, portanto, rodar o mesmo software em três transputers em duas unidades de tempo. Como P1 usa quatro unidades de tempo para rodar, ele terá de ser subdividido. Se P1 consiste em um conjunto de processos seqüenciais, que podem ser primitivos ou complexos, ele pode ser dividido em dois processos, P1a e P1b, que usariam aproximadamente o mesmo tempo para executar. Esses dois processos podem agora ser canalizados juntos e alocados em dois processadores disponíveis. Os processos occam P2 e P3 serão então alocados em um terceiro processador, onde eles podem executar conceitualmente em paralelo e ser escalonados pelo hardware escalonador do transputer. Naturalmente, como esta é uma solução canalizada, ela só executará eficientemente se forem processados conjuntos de itens de dados.

Tem sido desenvolvida uma técnica nos últimos anos que implementa canalização de duas dimensões: isto é conhecido como *processamento sistólico*. Processamento sistólico é ideal para implementação de vetores bidimensionais de transputers, onde cada transputer é conectado aos seus quatro vizinhos mais próximos via seus enlaces seriais. A Figura 4.10 mostra um vetor de transputers com canalização executando do topo para baixo e da esquerda para direita. Os processos em cada transputer lêem dados do canal esquerdo e do canal do topo, processam e passam os dados ou o resultado intermediário para os canais da direita e de baixo. Tal vetor poderia ser usado para multiplicação matricial, onde as duas entradas são multiplicadas juntas e somadas ao total:

```
SEQ
    PAR
        cima     ? x
        esquerda ? y
    PAR
        total := total + (x * y)
        baixo ! x
        direita ! y
```

No código occam acima, x, y e "total" foram declaradas como variáveis, com "total" iniciado com zero. "Cima", "baixo", "esquerda" e "direita" foram declaradas como canais. Será visto que x e y são saídas em paralelo com a avaliação da expressão associada a "total". Deste modo, processos vizinhos abaixo e à direita não são mantidos na espera desnecessariamente. Foi colocado anteriormente que os componentes de uma construção paralela, implementados em um processador simples, irão executar "conceitualmente" somente em paralelo. Entretanto, nesse caso particular, os dois processos de saída são manipulados por interfaces de enlaces que são capazes de operar independentemente do processador principal. Logo que o processador tenha iniciado as duas saídas, a comunicação com os transputers vizinhos procede em paralelo com a avaliação da expressão. Processos de comunicação paralela terão realmente um efeito leve no desempenho total do processador. A Inmos afirma que a performance dos processos não envolvidos diretamente com a comunicação será reduzida somente em 8%, mesmo quando os enlaces são operados no pico de suas taxas.

A Figura 4.10 mostra como os dados injetados na canalização em duas dimensões criam frentes (wavefront) diagonais de itens de dados. Neste exemplo, as frentes passam pelo vetor de processador do topo esquerdo para baixo e direita. Devido a essa analogia, as manipulações de matrizes implementadas dessa forma são referidas como *processamento de frente*. O nome sistolítico é derivado da analogia entre o pulso de dado através do vetor e a batida sistolítica do coração. Embora processos sistolíticos não sejam usualmente apropriados para problemas de software em geral, para aplicações matemáticas particulares eles podem ser muito eficientes.

4.9 PROJETO DE SISTEMAS CONCORRENTES

Qualquer aplicação que presentemente rode em um vetor convencional de vetores pode usualmente ser implementada em um vetor de transputers. Alguns trabalhos excelentes têm sido conduzidos na área de processamento de *wavefront* e processamento sistólico (particularmente por S. Y. Kung, que originalmente adotou a palavra "sistólico"),

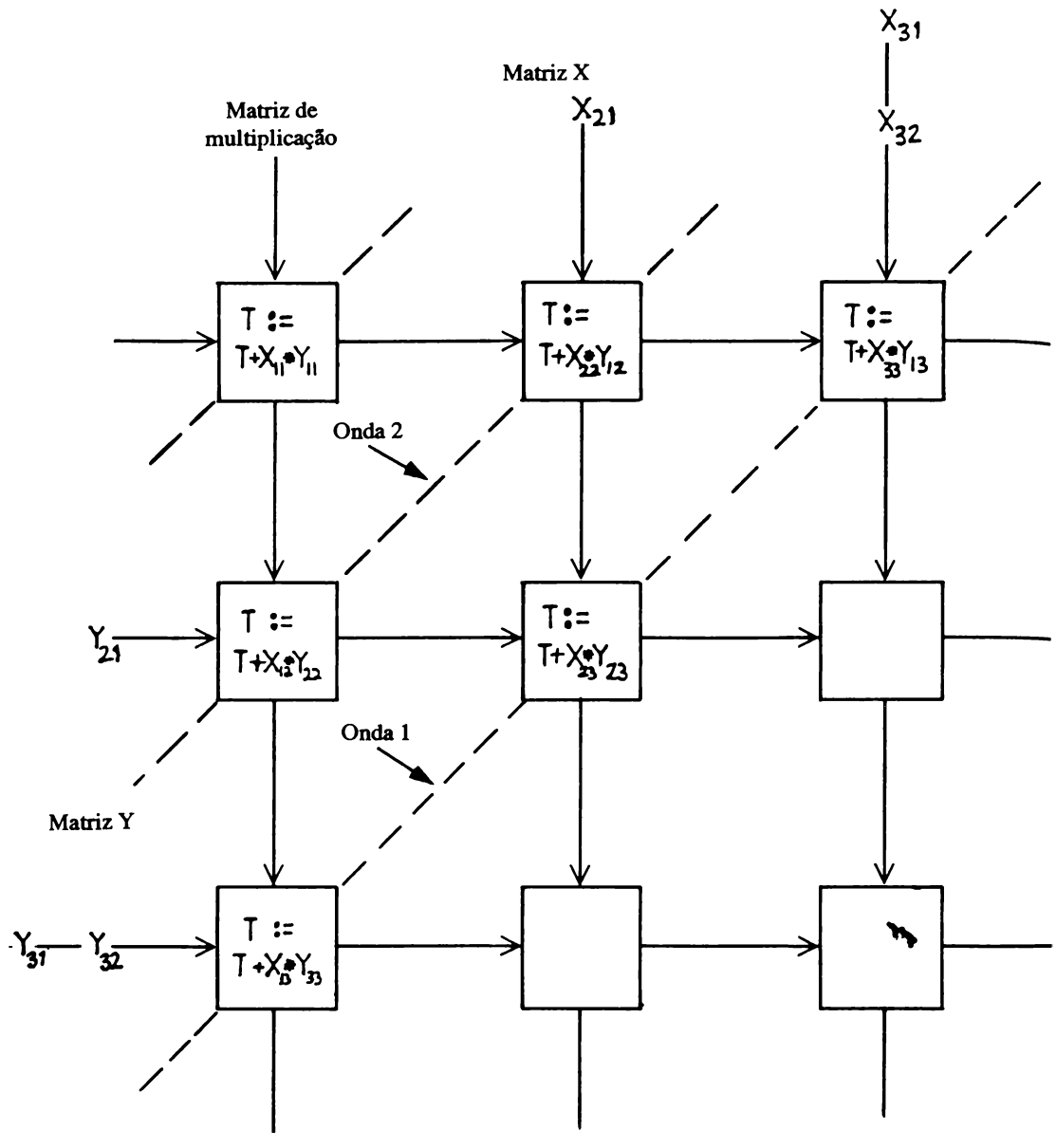


Figura 4.10 Processamento sistólico e de onda.

e essas técnicas são particularmente boas para implementação de vetores bidimensionais de transputers. Entretanto, sistemas de desenvolvimento para transputers deveriam não ser entendidos como um meio de transformar toda a aplicação em processo sistólico para um vetor bidimensional de processadores.

Projetos de sistemas com transputers apresentam um nível totalmente diferente de problemas no qual a maioria dos projetistas não está acostumada. No passo, inventar mecanismos para comunicação interprocessadores, escalonar tarefas e sincronização de processos tomava uma grande parte do tempo, dos esforços e da habilidade criativa do projetista. No transputer, a maioria desses problemas é manipulada pela linguagem occam e pelo próprio transputer. Livre da necessidade de considerar esses problemas, o projetista de sistemas pode encaminhar-se para problemas particulares associados com concorrência e arquitetura de sistemas grandes.

O projeto de sistemas concorrentes será uma experiência totalmente nova para a maioria dos projetistas, simplesmente porque máquinas concorrentes têm sido uma raridade até recentemente. O projeto de um programa seqüencial, rodando em uma máquina convencional do tipo Von Neumann, pode ser comparado a uma tarefa para um engenheiro projetista que tenha de considerar uma única linha de produção de uma fábrica. Ele deve garantir que todos os trabalhadores especialistas, máquinas e recursos estejam disponíveis para processar itens naquela única linha de produção. Tomando-se uma analogia similar, o projetista de um sistema de computador concorrente é como um projeto de uma fábrica completa. Aqui o projetista deve considerar não só todos os trabalhadores especialistas, máquinas e recursos requisitados para todas as linhas de produção da fábrica, mas também provê-los no número certo. Se máquinas particulares, recursos ou habilidades devem ser compartilhadas entre as linhas de produção, então o projetista deve garantir que não haverá bloqueios. Mais importante, ele deve permitir que os recursos críticos compartilháveis sejam monopolizados de tal forma que toda a fábrica trabalhe pesado para evitar uma paralisação. A complexidade dessa tarefa é tal que o projetista pode não ser hábil para resolver todos os problemas no quadro colocado. Alguns problemas podem tornar-se aparentes depois que a fábrica estiver em operação. Da mesma forma, sistemas de computadores concorrentes sempre requererão algum grau de depuração antes de serem completamente funcionais.

Ao lado da consideração de problemas especiais, vários projetistas de sistemas de transputers também terão de encaminhar problemas com projetos de vetores de processadores extremamente grandes. Alguns desses problemas serão considerados na próxima seção.

4.10 ARQUITETURA DE SISTEMAS

Para projetos convencionais, algumas das arquiteturas mais esotéricas colocadas em relação ao transputer podem parecer desnecessariamente complexas. De fato, a reação de muitos a tais idéias como dimensões de hipercubos é classificá-los no campo da ficção

científica em vez da área de projeto de sistema prático. Há, entretanto, algumas razões reais para o uso dessas arquiteturas em sistemas reais.

Os transputers provêem um meio de distribuição de software através de um número de processadores, mas deveria ser lembrado ainda que o custo adicional de comunicação sempre existirá, particularmente quando a comunicação precisa ser feita entre mais de um transputer. Como política geral o projetista deveria requerer o mínimo possível de comunicação interprocessador. Sempre que possível, certos grupos de software altamente acoplados deveriam residir no mesmo processador. Os grupos são identificados pela análise da representação do fluxo de dados do projeto de software. Os processos conectados por fluxo de dados com grande freqüência ou em grande volume são ditos como "altamente acoplados". Esse grupo de processos pode formar tipicamente um subsistema embutido em um sistema de computador, tal como um piloto automático em uma aeronave, ou uma seção de um algoritmo de processamento de sinal, precisando de uma quantidade relativamente pequena de dados e devolvendo resultados. Os estudos de desempenho indicam que grupos de processos emparelhados terão de rodar em mais de um processador; então, para reduzir-se o custo adicional de comunicação, todo processador que suporte este subsistema deveria ser conectado diretamente a todos os outros processadores que suportam aquela mesma tarefa. Essa configuração será familiar a qualquer analista de redes locais. Se n processadores são necessários para produzir o desempenho para o subsistema, então, se possível, cada processador deve ter n-1 enlaces para conectar-se diretamente com todos os outros processadores e um enlace para conectar-se com outro subsistema ou com o mundo externo. Isto faz um total de (n-1) + 1 = n enlaces necessários para cada processador no subsistema. Portanto, o número de processadores que podem ser ligados na rede para formar um grupo é igual ao número de enlaces que cada um tem disponível. O transputer T424 pode, portanto, ser enlaçado em grupos de quatro (veja Figura 4.11).

Um estudante de química, ou qualquer um que tenha usado um kit de modelo molecular, saberá que, conectando objetos a seus vizinhos mais próximos por quatro enlaces regulares, forma-se uma estrutura de dimensão tetraedral (veja Figura 4.12). Essa estrutura é a que dá a um cristal de diamante sua rigidez. Também é a forma ideal de conectar uma rede grande de transputers de quatro enlaces para diminuir o custo de comunicação. O problema com tal rede é que ela não se presta a ser implantada em um circuito impresso bidimensional. A Figura 4.13 é uma representação tridimensional da Figura 4.14. Pode-se ver que é diferente da estrutura do diamante na Figura 4.12, na qual ele não é totalmente regular, mas formado por grupos de quatro processadores cada. Embora essa estrutura de rede tenha sido alcançada por um método puramente lógico, ainda é difícil acreditar que essa estrutura provenha um custo de comunicação mais baixo que o da conexão norte-sul/leste-oeste de vetores, mostrada na Figura 4.13. Uma verificação rápida,

entretanto, usando-se o vetor da Figura 4.4, mostra que um sinal passando de um transputer na ponta de um vetor convencional para um transputer na ponta oposta tem que passar por, no mínimo, seis enlaces. No vetor mostrado na Figura 4.14, o sinal precisa passar por três enlaces para os processadores nas mesmas posições relativas. De fato, três é o número máximo de enlaces usados para conectar dois processadores em uma rede.

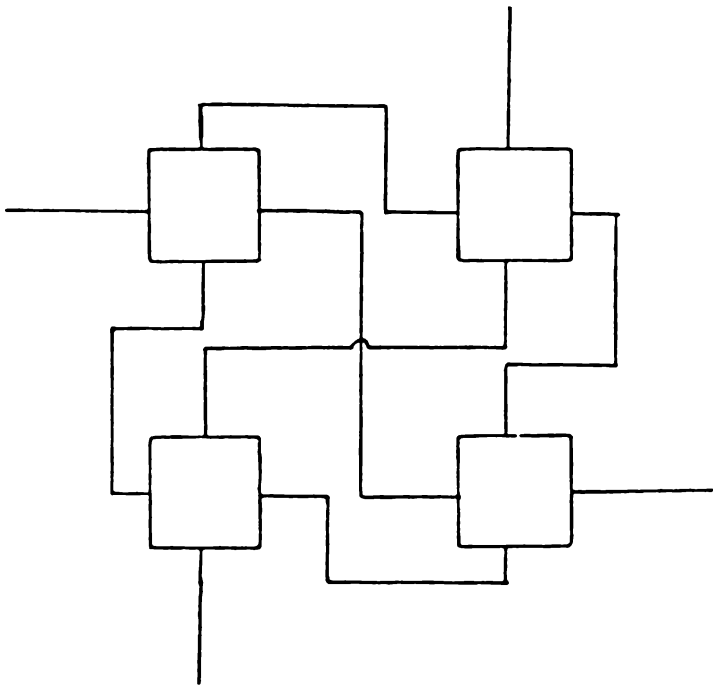


Figura 4.11 Quatro transputers interconectados.

Seria tentador repetir este exemplo para níveis superiores, construindo-se redes de 64, depois 256 transputers, e assim por diante. Infelizmente a fiação poderia tornar-se rapidamente um problema, e os enlaces interprocessadores poderiam exceder o limite de 400 mm. Entretanto 16 transputers formam um tamanho bastante prático de grupo. Se 16 T424s puderem ser colocados em uma única placa de circuito impresso, os grupos tornar-se-ão ainda mais práticos.

Redes de duas dimensões de transputers nos quais os processadores são conectados a seus quatro vizinhos mais próximos, como mostrado na Figura 4.4, podem de fato ser úteis para implementação de algoritmos que envolvem manipulação de matrizes. Este vetor é também aplicável a várias tarefas de processamento de imagens onde os transputers podem ser usados tanto para processar uma imagem de entrada como para gráficos

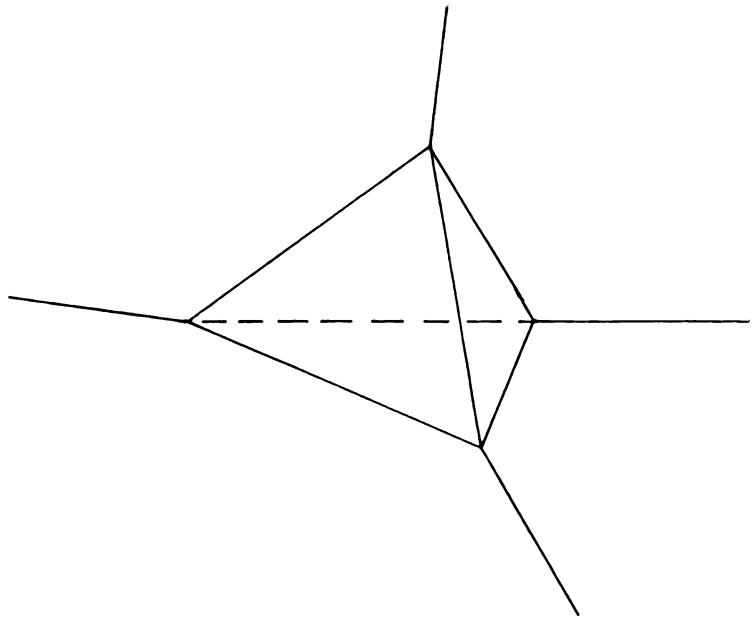


Figura 4.12 Rede conectada em estrutura de diamante.

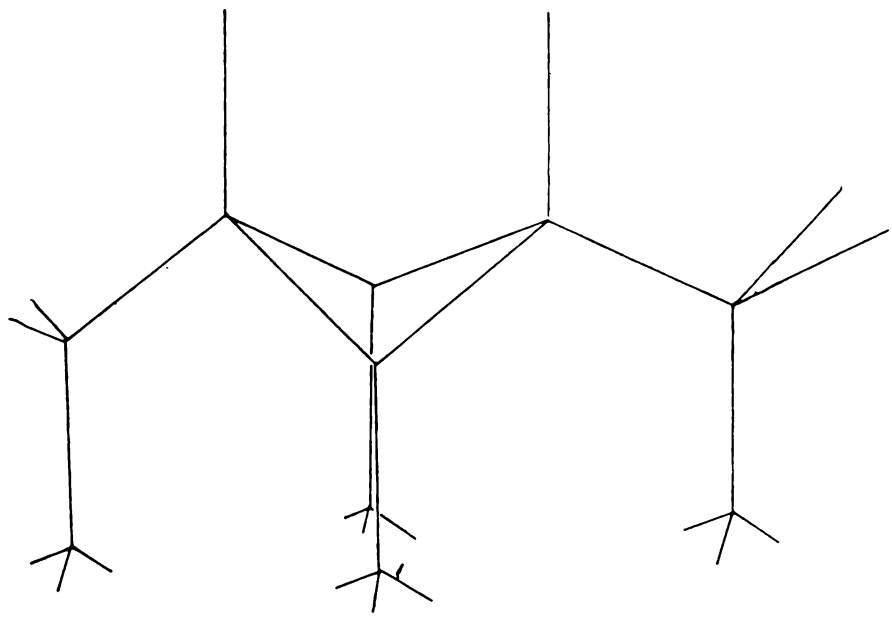


Figura 4.13 Representação tridimensional das conexões da Figura 4.11.

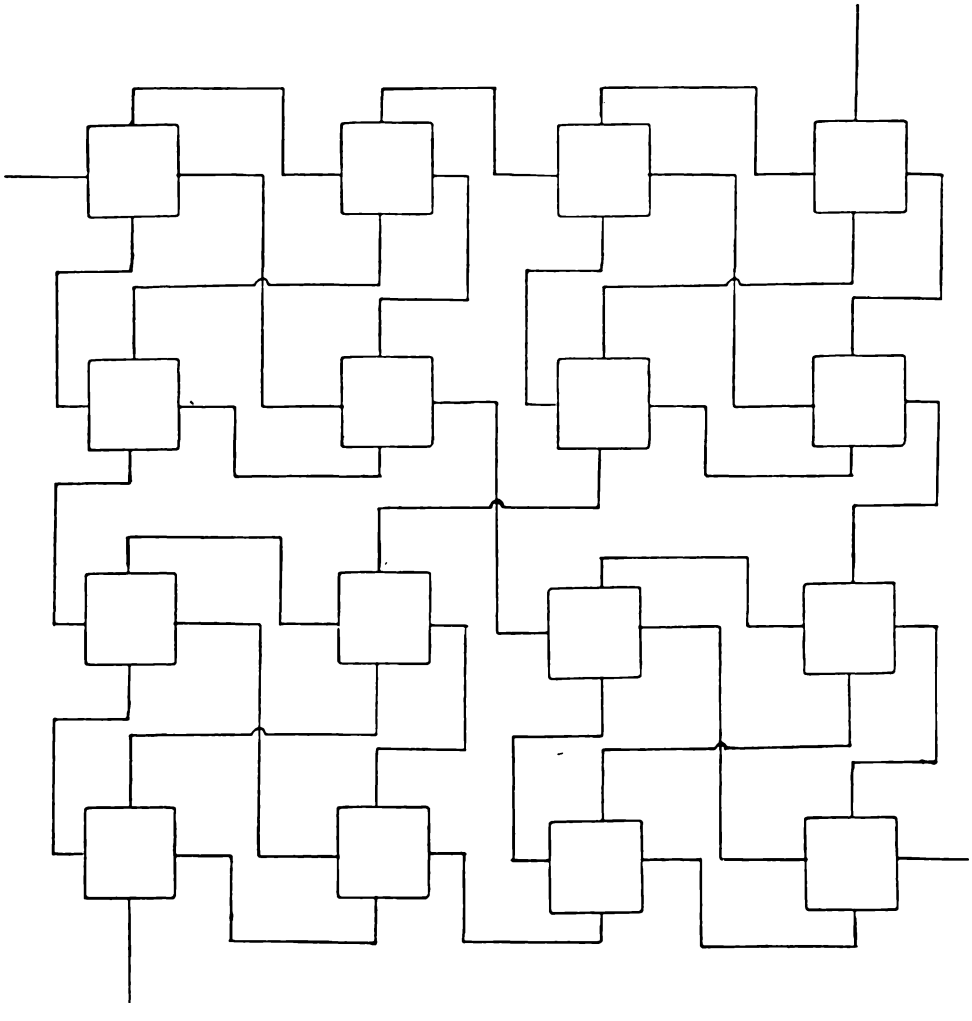


Figura 4.14 Dezesseis transputers interconectados.

gerais, com cada processador sendo alocado a um número de pixels para controle. Alguns algoritmos de processamento de imagens requerem que cada processo modelado comunique-se com oito vizinhos em vez de quatro. Isto não requer uma arquitetura particularmente novelesca, apenas seria preferível que cada processador tivesse oito enlaces em vez de quatro, de forma que todas as diagonais poderiam ser conectadas.

Unindo dois transputers e usando um enlace de cada, produz-se um processador dual com seis enlaces disponíveis:

$$(2\times4)-2=6$$

Tendo seis enlaces permite-se que redes locais cúbicas de transputers sejam construídas em três dimensões. Conectando três transputers em uma estrutura linear provê-se um processador triplo com oito enlaces. Isto permite que se construa uma rede de quatro dimensões. Hipercubos multidimensionais não são estruturas de rede imaginárias, elas podem realmente ser construídas. Entretanto os problemas de se tentar construir um quadro de circuito desenhado com multinfveis tornam-se um pesadelo. Em teoria, um hipercubo de dimensão n pode ser construído por meio e cubos contendo n-1 transputers com 2n enlaces cada, onde n pode ser qualquer número. Na prática parece ser de melhor senso que se espere até que os transputers com seis, oito ou 10 enlaces sejam fabricados. Construir-se redes de tridimensionais é uma consideração válida, particularmente onde processos tridimensionais estão sendo modelados, mas a viabilidade da construção de hipercubos deve ser questionada. Pode-se encontrar usos melhores para hipercubos com lados menores, como supergrupos para hospedar um subsistema altamente acoplado. Por exemplo, um cubo de quatro dimensões de lado 2 poderia conter $2^4 = 16$ nós, com cada nó contendo três transputers. Tal supergrupo, ou mini-hipercubo, pode encontrar uso prático, mas interconectar-se 48 transputers em tal arquitetura complexa não é uma tarefa fácil.

O argumento para hipercubos é que eles reduzem a latência de comunicação. Considere-se duas redes locais, cada uma contendo 48 transputers; um vetor de duas dimensões de 6×8 e um hipercubo triplo com nós de transputers 2×2×2×2. Tomando o pior caso na tentativa de comunicar os transputers com a extremidade oposta de cada rede, no vetor de duas dimensões, a mensagem é reposta por onze transputers, na rede de quatro dimensões ela pode ser reposta por qualquer coisa entre dois a dez transputers, dependendo da orientação de cada nó. Na média, portanto, comunicação entre hipercubos é mais rápida. Previsão de impasse em tal estrutura, entretanto, será impossfvel até que um analisador de impasse automático seja desenvolvido.

4.11 DESENVOLVIMENTOS FUTUROS

Foi visto que a arquitetura geral do transputer permite que toda uma família de componentes seja desenvolvida, cada uma tendo sua própria área de aplicação. A característica única do transputer oferece várias outras possibilidades excitantes. Vários projetos de vetores de duas dimensões de transputers se parecem bem similares ao modo que as pastilhas são dispostas na fatia de silício no qual são fabricadas. Parece um grande desperdício de tempo, esforço e dinheiro cortar essas pastilhas, encapsular as pastilhas individuais, projetar e fabricar um circuito impresso para que as pastilhas possam ser interconectadas no mesmo circuito no qual elas foram manufaturadas. Seria de mais senso co-

nectar-se os transputers a seus vizinhos enquanto eles estão ainda em pastilhas. A integração na escala de bolacha, como isto é conhecido, já foi tentada com dispositivos de memória, que como os transputers podem ser conectados em redes regulares.

Integração a nível de bolacha tem dois grandes problemas a serem mencionados. Primeiro, a garantia de produzir pastilhas 100% operantes é rara. Simplesmente conectar-se o transputer a todos os seus vizinhos inevitavelmente incluiria falta de transputers na rede. Segundo, existem os problemas de alimentação e dissipação. A alimentação deve chegar a cada transputer, e um watt de calor gerado por cada transputer deve ser removido.

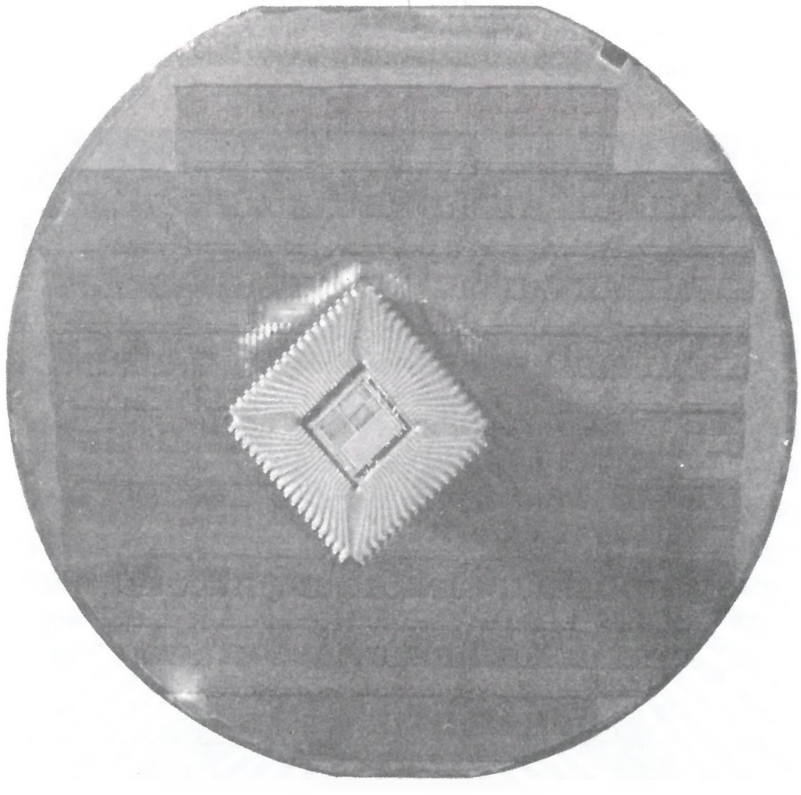


Figura 4.15 O transputer T414 em bolacha (*Cortesia do grupo de companhias Inmos*).

Obviamente, em primeiro lugar, todos os esforços devem ser feitos para aperfeiçoar-se as técnicas de produção para alcançar o máximo possível de produção de pastilhas operantes. Em termos maiores, entretanto, um software poderia ser desenvolvido para configurar os vetores para contornar a falta de pastilhas. Se o software pudesse configu-

rar dinamicamente as redes e monitorar continuamente a condição de cada pastilha, as pastilhas que falhassem em operação poderiam ser descartados também. A solução a longo prazo para o problema de alimentação é desenvolver-se novos semicondutores que requeiram menos energia. Com um bônus somado, esses materiais podem apresentar uma impedância baixa que poderia permitir que os sinais fossem passados através das pastilhas mais rapidamente, portanto incrementando o desempenho de cada processador individual.

A solução a curto prazo para a integração de escala de pastilhas de transputers seria identificar todas as ocorrências de vetores dois-por-dois de pastilhas funcionando na bolacha. Esse processo poderia ser suportado por um dispositivo auxiliar de computador que tenha sido alimentado com as posições de todas as pastilhas em funcionamento por um equipamento de teste automático. Dividindo a pastilha para cortar seqüências irregulares de vetores dois-por-dois pode ser mais difícil, mas é possível. É possível também encapsular as pastilhas funcionais nesses vetores dois-por-dois.

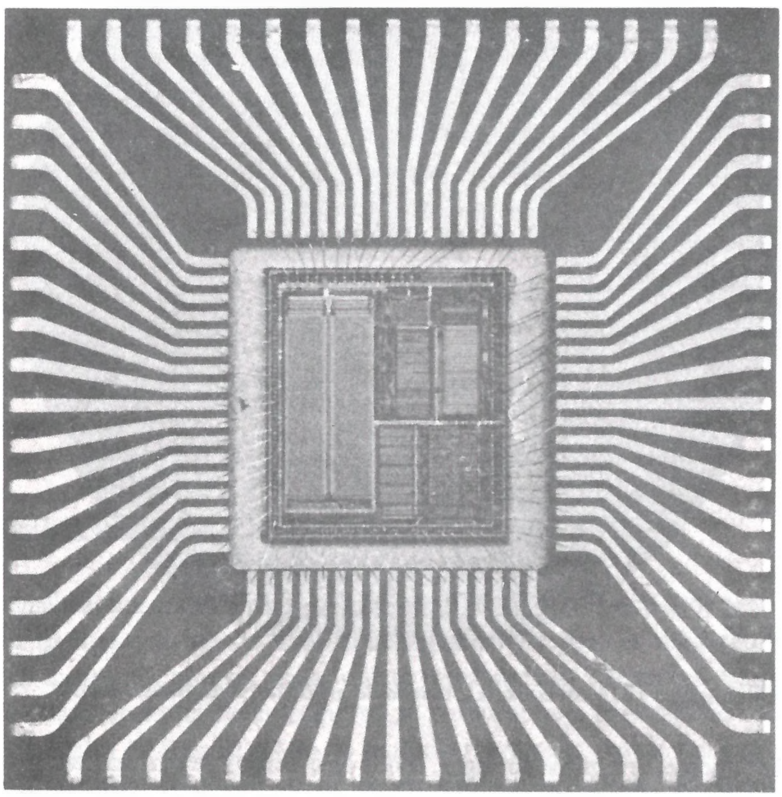


Figura 4.16 O transputer T414 (*Cortesia do grupo de companhias Inmos*).

Novamente a longo prazo, a conexão de processadores na bolacha seria provavelmente uma forma de enlace paralelo, provavelmente com largura de 32 bits, o que poderia aumentar consideravelmente a largura da banda interna da pastilha. Para os primeiros produtos transputer de bolacha, os enlaces seriais padrão forneceriam um meio simples de conectar processadores, sem a necessidade de modificar a arquitetura interna.

Tendo vetores de transputers dois-por-dois em cada pastilha poderíamos reduzir os problemas de necessidade de energia, em cada processador, assim como todos os quatro seriam acessíveis pela borda da pastilha. O calor dissipado poderia apresentar maiores dificuldades e alguma forma de resfriamento da pastilha seria necessária, possivelmente integrando-se canalização de calor e dissipadores de calor dentro da pastilha. Como esses problemas são gradualmente solucionáveis, será possível produzir vetores maiores de transputers integrados. Uma pastilha de 10 cm quadrados, usando a tecnologia de 1985, poderia operar a um bilhão de operações por segundo e também incorporar 100 kbytes de memória rápida.

Para aqueles projetos que necessitam de mais de quatro enlaces em seus transputers, a bolacha integrada de quatro T424 pode ser a resposta para o problema. Em adição ao desempenho de 40 MIPS, um quad T424 teria oito enlaces seriais com uma banda total de 80 Mbits/s. Em adição às vantagens de desempenho, pode realmente ser que o quad T424 possa ser desenvolvido em uma escala de tempo menor que um transputer simples com oito enlaces.

Poderia ser perfeitamente claro a este ponto que essas idéias para produtos futuros, no momento, são pura especulação. Várias representações da Imnos têm, através do último ano, mencionado a possibilidade de produtos em escala de bolacha, mas não há um compromisso real para desenvolver esses dispositivos. O desenvolvimento deles, como aquele dispositivo individual na família de transputers, dependerá da demanda, investimento em pesquisas e desenvolvimento e da habilidade de contornar o grande número de problemas técnicos que serão encontrados inevitavelmente.

O que é certo é que como o transputer é usado em mais e mais aplicações reais, a necessidade de muitas variações de dispositivos irá surgir. O transputer é o desenvolvimento mais significativo desde o microprocessador. Esse dispositivo do tamanho de um microprocessador, com a força de um computador de grande porte, fornece-nos um bloco de construção para substituir a maioria das máquinas atuais. Processadores de propósito geral, processadores vetorizados, supercomputadores e máquinas que não tinham sido possíveis até agora, como as máquinas de quinta geração, podem todos ser montados por transputers. Mais importante, o transputer existe aqui e agora. No desenvolvimento de produtos novos a partir deste dispositivo, somos limitados apenas por nossa imaginação.

REFERÊNCIAS BIBLIOGRÁFICA

1. *The Occam Programming Manual,* Prentice-Hall Int., 1982.
2. *Towards Fifth-Generation computers,* Simons, G.L., NCC Publications, 1983.
3. *Transputer Reference Manual,* Inmos, 1985.
4. Kung, S.Y., IEEE ASSP magazine, July, 1985.
5. *System Development,* Jackson, M.A., Prentice-Hall Int., 1983.

CAPÍTULO 5

O 80386 DA INTEL

Nick John
Gerente de Marketing
Grupo de Microcomputadores, Intel Internacional

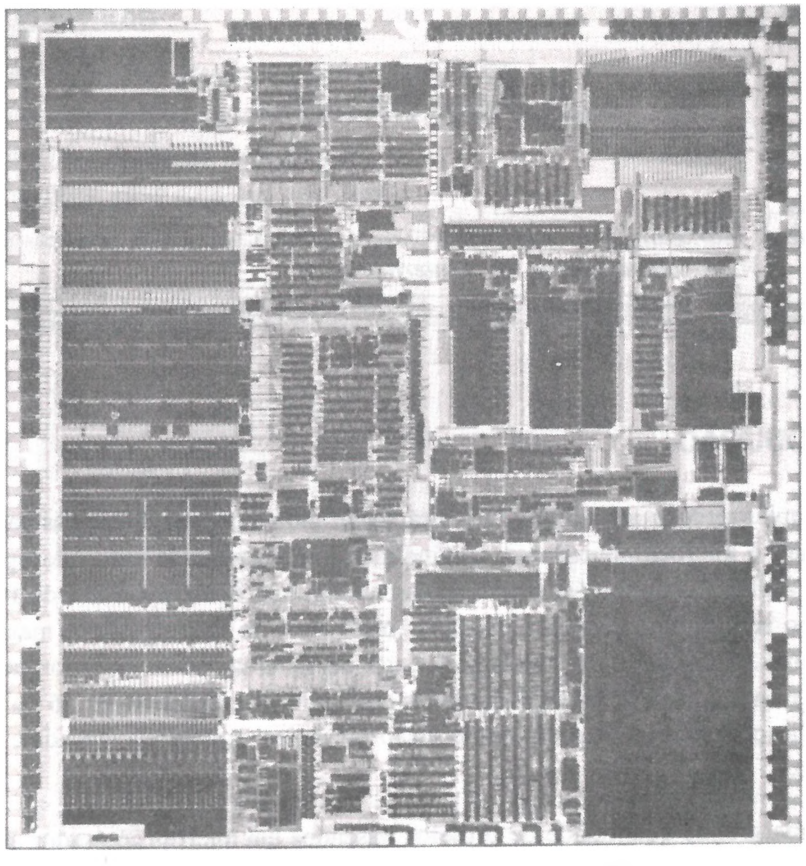
5.1 INTRODUÇÃO

O microprocessador de 32 bits está acelerando a convergência das bases de aplicação de mini e microcomputadores, originalmente incrementada pelo microprocessador de 16 bits. Muitas das aplicações que hoje demandam microprocessadores de 16 bits estão evoluindo no sentido de necessitar de níveis mais altos de desempenho e mais espaço de endereçamento. Além disso, os atuais projetistas de sistemas estão incorporando características que nunca antes estiveram disponíveis em microprocessadores de 16 ou 32 bits, em áreas de aplicações emergentes tais como visão mecânica, robótica avançada e reconhecimento de voz.

Uma grande quantidade de projetos baseados em microprocessador alcançou um estágio onde o fator software é o componente principal do custo geral de desenvolvimento, bem como um fator-chave para trazer benefícios de produtos ao mercado. Os sistemas comerciais multiusuários de amanhã e as estações de trabalho multitarefas, para engenharia requerem um alto nível de sofisticação de software para serem efetivos.

A importância crítica do software impõe condições correspondentes nos microprocessadores de 16 e 32 bits.

Inicialmente um microprocessador de 32 bits deve permitir o desenvolvimento de compiladores altamente eficientes, de tal forma que os sistemas operacionais e aplicações possam espelhar o desempenho do microprocessador. O microprocessador deve também oferecer um conjunto de características estendidas para dar suporte às necessidades do projetista de sistemas. Aplicações tais como transações financeiras precisam oferecer a possibilidade de proteção de dados confidenciais. Estações de trabalho e sistemas comer-



Com mais de 275.000 transistores e um processo CHMOS-III de 1,5 μm, o microcomputador 80386 da Intel combina um processador de 32 bits e funções de gerenciamento de memória em uma única pastilha.

ciais de alto nível requerem amplos espaços de endereçamento lógico para cada programa individual. Finalmente, em todas as áreas de aplicação, a capacidade de permitir a migração do software existente e compatibilidade com microprocessadores anteriores é de altíssima importância.

O poderoso conjunto de instruções, alta performance e flexibilidade do 80386 da Intel fazem-no uma escolha superior para uma ampla gama de ambientes de execução, tais como sistemas de chaveamento de escritório central, controladores industriais e estações de trabalho para engenharia.

Neste capítulo trataremos em detalhe o 80386, especialmente nas implicações pertinentes a projetistas de software e de sistemas.

5.2 MODELO DE PROGRAMAÇÃO

Proveniente diretamente dos requerimentos de desempenho de software avançado, um dos objetivos primários no projeto do 80386 foi torná-lo um excelente alvo para geração de código. Dois aspectos importantes da máquina auxiliam no alcance desse objetivo. Um é o conjunto completo de registradores de 32 bits que podem ser usados, sem restrição, tanto para executar cálculos como para compor endereços de memória. O outro é o rico conjunto de tipos de dados suportados pela máquina.

5.2.1 CONJUNTO DE REGISTRADORES

O 80386 possui diversos conjuntos de registradores internos para dar suporte a características avançadas. A Figura 5.1 ilustra esses conjuntos.

Oito registradores de uso geral (EAX, EBX, ECX, EDX, ESP, EBP, ESI e EDI) são usados para executar cálculos e endereçamento de memória; isto provê um ótimo conjunto de registradores para implementar todos os modos de endereçamento mais comuns, bem como um amplo conjunto de variáveis tipo registrador, sem gastar silício em registradores desnecessários. Três *registradores de controle* e um de *indicadores* controlam o comportamento da máquina e reportam o estado de diversas operações. Seis registradores adicionais (CX, SS, DS, ES, FS e GS) podem ser usados para estruturar o amplo espaço de endereçamento de 64 terabytes (trilhões de bytes) em espaços lógicos separados denominados *segmentos;* cada programa pode então possuir seis espaços de endereçamento lógico mapeados a um só tempo, incluindo quatro espaços de dados separados. Além disso, seis *registradores de depuração* controlam a sinalização de quatro *breakpoints de dados* ou *código* para prover capacidades de depuração anteriormente não disponíveis.

5.2.1.1 OPERANDOS DE ENDEREÇAMENTO

Provavelmente o método mais simples de armazenar um operando é num registrador. Por exemplo, para executar o cálculo:

$$\text{potência} = \text{voltagem} \times \text{corrente},$$

a seguinte seqüência de instruções poderia ser usada, assumindo que a corrente já está armazenada em EBX, a voltagem em ESI e a potência deve ser guardada em EDI:

```
MOV EDI,ESI : guarda a voltagem
IMUL EDI,EBX : potência = EDI × corrente
```

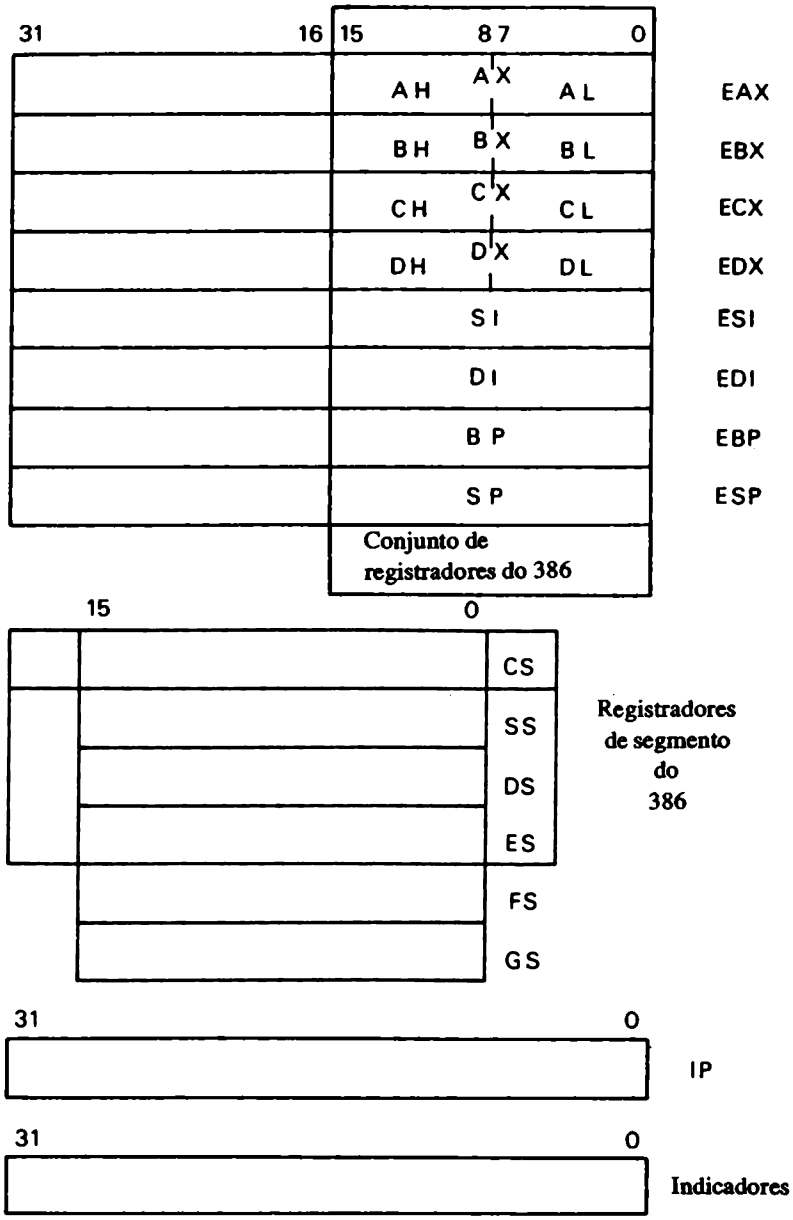


Figura 5.1 Arquitetura base dos registradores do 80386.

Esse esquema é bastante efetivo quando variáveis tipo registrador são desejadas (como em linguagem C) do ponto de vista do projetista de compilador. As qualidades ortogonais dos modos de endereçamento do 80386 permitem múltiplas variáveis tipo registrador.

Depois do registrador, a fonte mais comum de um operando é a memória. Em sua forma mais simples, o endereço efetivo, ou indexador de byte, de um operando dentro de um certo espaço de endereçamento lógico pode ser diretamente especificado por uma instrução. No entanto, muitas vezes um endereço particular de memória não é conhecido até que o programa esteja realmente executando, tal como em acessos a matrizes dinâmicas e execução de código relocável. Nesses casos, o endereço efetivo pode ser formado pela soma dos conteúdos de um ou, opcionalmente, dois registradores de uso geral e um valor ótimo de constante ou deslocamento (*displacement*).

A geração de endereço efetivo baseada em registrador pode ser resumida como se segue:

[registrador de base] + [registrador indexador]*[escala] + [deslocamento]

Aqui o registrador de base é qualquer registrador de uso geral, e o registrador indexador é qualquer um menos ESP. Desde que ESP sempre aponta a pilha corrente, ele pode ser facilmente usado para endereçamento relativo à pilha. A escala é um valor constante, 1, 2, 4 ou 8, pelo qual o indexador pode ser multiplicado. O campo de deslocamento é também uma constante: seu valor pode ser qualquer um na faixa -2^{31} a $2^{31}-1$. A Tabela 5.1 detalha os modos de endereçamento de 32 bits do 80386.

Desde que base, indexador e deslocamento são opcionais, e a escala também é opcional quando um indexador está presente, o 80386 oferece onze modos de endereçamento de 32 bits distintos. Esse conjunto de modos inclui todos aqueles comumente usados em compiladores de linguagens de alto nível. Vamos considerar o uso de alguns desses modos.

Assuma que parâmetros são passados para uma sub-rotina através da pilha. Então o último parâmetro poderia ser colocado na pilha do registrador EAX usando o modo base deslocamento, especificando:

```
MOV DWORD PTR [ESP+4],EAX ; 2 ciclos de relógios
```

Similarmente, assuma que o ponteiro Next Record está armazenado no registrador EAX, o comando:

```
Next Record = Next Record.Link;
```

poderia ser traduzido como:

MOV EAX, DWORD PRT[EAX+LinkOffset]; 4 ciclos de relógios

Tabela 5.1 Modo de endereçamento do 386

Base	*Índice*		*Deslocamento*	
[EAX ECX EDX EBX ESP EBP ESI EDI]	+ [EAX ECX EDX EBX EBP ESI EDI]	* [1 2 4 8]	+ [0 / 8-bit deslocamento / 32-bit deslocamento]	
			MODO	
			Desloc	Apenas deslocamento
Base				Base
Base			+Desloc	Base e deslocamento
	Índice			Índice
	Índice	* Escala		Índice com escala
	Índice		+Desloc	Índice e deslocamento
	Índice	* Escala	+Desloc	Índice com escala e deslocamento
Base	+Índice			Base e índice
Base	+Índice	* Escala		Base e índice com escala
Base	+Índice		Desloc	Base com índice e deslocamento
Base	+Índice	* Escala	Desloc	Base com índice, escala e deslocamento
Registrador				Operando registrador
Imediato				Operando imediato

Isto permite um deslocamento numa estrutura a ser usado sem custo adicional. Finalmente, vamos assumir que V aponta para um vetor inteiro dinamicamente alocado e que vamos iniciar v[i+2] com a variável armazenada em EDI, o comando:

MOV DWORD PTR[ESI+EDI*4+8],5; 3 ciclos de relógios

usa o modo base indexado com escala e deslocamento e executa a iniciação desejada. O uso de indexação com escala para gerar um índice de um byte numa matriz inteira (4 bytes) elimina a necessidade de outra instrução para gerar o byte índice. Adicionalmente, todos os modos de endereçamento da família 8086 de 16 bits são suportados pelo 80386 para manter compatibilidade com os membros de 16 bits da família 8086 (8086, 8088 e 80286).

Note o número total de ciclos de relógios para cada instrução. Uma razão fundamental para o alto desempenho de 80386 é que a formação de endereço efetivo para uma dada instrução é iniciada durante o último relógio da instrução anterior, quando esta executa a linha de código. Desde que ele utiliza apenas um relógio para formar o endereço efetivo quando não há um índice especificado, essa formação é quase sempre ocultada. E, em virtude da Unidade de Gerenciamento de Memória (MMU) interna à pastilha, a formação do endereço efetivo inclui a tradução de endereço lógico em físico via as unidades de segmentação e paginação. Assim, virtualmente não há atraso de geração e tradução de endereço no 386. A única exceção ocorre quando um indexador é usado. Para esse caso, dois relógios são necessários para a geração do endereço efetivo, resultando num atraso efetivo de um relógio.

5.2.2 TIPOS DE DADOS

O suporte a tipos de dados no 8386 é direcionado para cobrir todos os tipos fundamentais encontrados na maioria das linguagens de alto nível. O 80386 permite operações básicas sobre todos esses tipos de dados. Eles estão ilustrados e descritos em mais detalhes na Figura 5.2.

	Ordinal	*Inteiro*	*BCD*	*Ponto flutuante*	*String*	*Bit string*
Movimento De/para memória Conversão de precisão	x	x	1	2	x	x
Aritmético Soma, subtração multiplicação, divisão negação	x	x	1	2		
Lógicos And, Or, Xor, deslocamento	x	x				
Comparação	x	x	1	2	x	x
Transcedentais				2		

1. Instruções de ajuste ASCII e decimal são fornecidas para implementos malhas eficientes. BCD é suportado diretamente pelos co-processadores numéricos 287 e 387.
2. Disponíveis utilizando diretamente os co-processadores numéricos 287 e 387.

Figura 5.2 Suporte de tipos de dados e conjunto de instruções.

5.2.2.1 ORDINAL

Um ordinal é um número não assinalado na faixa de 0 a 4294967295 ($2^{32}-1$). Como com inteiros, existem variáveis de ordinais de 32, 16 e 8 bits.

5.2.2.2 PONTEIRO

Um ponteiro descreve um endereço de memória. Todos os ponteiros possuem dois componentes: um seletor de 16 bits, que dá nome ao espaço de endereçamento lógico, e *um endereço efetivo* ou *deslocamento* de 32 bits, que especifica o índice de byte dentro do espaço de endereçamento lógico. Desde que todos os endereços podem ser gerados usando um seletor implícito, o par completo seletor/deslocamento de 48 bits pode muitas vezes ser abreviado para o deslocamento de 32 bits, o qual pode endereçar um espaço linear de endereçamento de 4 gigabytes. Uma versão compacta apresenta um seletor de 16 bits e um deslocamento de 16 bits, que também dá suporte à compatibilidade de software com a família iAPX86.

5.2.2.3 INTEIRO

Um inteiro é um número assinalado em complemento de 2 na faixa –2147483648 a 2147483647 (-2^{31} a $2^{31}-1$). Para números que não possuem uma magnitude tão grande, também há suporte para inteiros assinalados de 8 e 16 bits. Um exemplo de operação com inteiro é a seqüência de instruções:

```
IMUL EBX, V[EDX*4] ; EBX = EBX*V[EDX] assinalada
```

Aqui, o conteúdo de EBX é multiplicado pelo EDXésimo elemento da matriz inteira V e o resultado é guardado em EBX.

5.2.2.4 BCD

Primitivas para auxiliar em aritmética decimal, tanto compactada como não compactada, são suportadas pelo 80386. Além disso, os processadores numéricos 287 e 387 provêem suporte direto para aritmética decimal compactada em 80 bits.

Agora considere o fragmento de programa em C:

```
struct A        Present : 1;
                DPL     : 2;
                Seg     : 1;
                ECRA    : 4;
            ç   accessRights,

regVAR = accessRights.DPL;
```

Assumindo que regVAR é uma variável tipo registrador armazenada em ESI, esse código pode ser traduzido diretamente em instruções do 80386, como a seqüência:

```
MOV CL,2                        ; extensão do campo de iniciação
MOV EAX,5                       ; deslocamento do campo de iniciação
XBITS ESI, accessRights, EAX, CL
```

5.2.2.5 PONTO FLUTUANTE

O 386 suporta ambos os co-processadores numéricos 80287 e 80387. Esses co-processadores estendem os tipos de dados suportados pelo 80386 incluindo inteiros longos de 64 bits, inteiros BCD de 18 dígitos, reais simples de 32 bits, reais duplos de 64 bits e reais estendidos de 80 bits. O 287 e 387 também tornam disponível um conjunto completo de instruções numéricas para permitir o uso desses tipos de dados.

5.2.2.6 OPERAÇÕES COM PRECISÃO MÚLTIPLA

O 80386 dá suporte a operações em 64 bits. Multiplicação e divisão de inteiro e ordinal são suportadas com produtos e dividendos de 64 bits. Adição e subtração em múltipla precisão também são facilmente implementáveis com o uso de operações soma com vai-um e subtração com vai-um do 80386.

Além de operadores aritméticos, o 80386 também suporta operadores de deslocamento de 64 bits com resultados de 32 bits. Essa operação é útil para criar malhas de alto desempenho para processamento de operandos baseados em memória não alinhados.

Como exemplo, vamos ver como se constrói uma rotina para movimentar um string de bits de uma posição para outra, como poderia ser necessário numa aplicação gráfica mapeada em bits. Assuma aqui que o registrador ESI armazena um endereço alinhado de palavra dupla, o registrador CL armazena o deslocamento de bit com relação ao início do string de bits e EDI possui o endereço de destino.

```
   LODS EAX,[ESI]; PEGA PRIMEIRA PALAVRA DO STRING, INCREMENTA
SRC                                    PTR
   MOV  EDX,EAX;          EDX = PRIMEIRA DWORD
   LODS  EAX, [ESI];  PEGA PRÓXIMA DWORD DO STRING, INCREMENTA

SRC                                    PTR
LOOP:  SHRD EAX,EDX,CL; ALINHA DWORD: EXD = EAX:EDX
                                                DESLOCADO DE

CL BITS

XCHG EDX,EAX;  TROCA EAX COM EDX
STOS  [EDI],EAX;  GUARDA DADOS ALINHADOS, INCREMENTA EDI
DEC   EBX;        DECREMENTA CONTADOR DE TRANSFERÊNCIA
JA    LOOP;       REPETE MALHA SE CONTADOR >0
```

Essa rotina assume que o destino é uma palavra dupla alinhada. Para permitir um campo de bits arbitrário para o destino, necessitamos apenas mover bits suficientes da fonte para o destino para alinhar o mesmo. Após isso ser feito, o código mostrado acima pode ser usado.

5.2.2.7 CAMPOS DE BITS

Seqüências contíguas de até 31 bits de extensão podem ser lidos/armazenados de/para um string de bits mais longo, que pode ter tamanho de até 4 gigabytes, e valores de bits simples podem ser testados e modificados. Um exemplo do uso de um campo de bits é a seqüência de instruções:

```
esperaLp:    LOCK BTS sema,2
             JC       esperaLp
```

Isto implementa um semáforo executando um teste e sinalização atômicos (trancados) no segundo bit do byte *sema* e esperando até que esse bit seja limpo.

5.2.3 TRADUÇÃO DE ENDEREÇO

O 80386 provê um mecanismo para endereçamento de memória tanto poderoso quanto completo. A maior contribuição para esse alto nível de funcionalidade é dada pela Unidade de Gerenciamento de Memória (MMU) interna à pastilha.

Começando com o programa do usuário, os endereços manipulados ao nível de conjunto de instruções são lógicos. Eles consistem em um par seletor/deslocamento – o seletor especifica um espaço de endereçamento lógico, o deslocamento especifica o byte dentro do espaço lógico. Desde que deslocamentos são quantidades de 32 bits no 80386, cada espaço de endereçamento lógico provê 32 bits de endereçamento linear. O mecanismo de segmentação do 80386 executa a busca em tabela e adição necessárias à tradução do par de endereço lógico visualizado por um programa num endereço linear.

Um endereço linear tem 32 bits de endereçamento. Esse intervalo de endereço é mapeado em endereços físicos através de mecanismos de paginação da MMU em pastilha. Endereços físicos são o que realmente aparecem nos pinos do barramento de endereços do 80386. A Figura 5.3 ilustra os dois níveis de tabelas de página que são usadas na tradução de endereço linear-para-físico no 80386. O endereço linear é dividido em três campos: *um indexador de tabela de diretório de página, um indexador de tabela de página* e *um indexador de byte de página*. O indexador de tabela de diretório de página é um campo de 10 bits que seleciona uma das 1024 tabelas de páginas. A tabela de página selecionada é então indexada pelo indexador de tabela de página (10 bits), portanto aponta para uma das 1024 páginas. A página real é um bloco de memória de 4096 bytes que finalmente é indexado pelo byte indexador de 12 bits. O byte assim alocado especifica o byte menos significativo do operando a ser endereçado.

As entradas das tabelas de segmentação são de 64 bits, chamadas *descritores de segmento*. Esses descritores contêm o endereço linear base do intervalo de endereço lógico, ou segmento, e outras informações sobre o intervalo de endereços. As entradas do diretório de páginas e tabelas de páginas são de 32 bits chamados *descritor de página* e armazena o endereço físico base da página junto a outras informações, tais como sujeira tradicional, bits presente e acessado para permitir a implementação de políticas de substituição de páginas por demanda. A informação contida nos descritores de segmento e página provê muitas outras características avançadas que são descritas na Seção 5.3.

Note que o mecanismo de paginação do 80386 é configurável. Se desejado, a paginação não precisa ser usada; no caso, o endereço de saída da unidade de segmentação é o endereço físico.

5.3 MODELO DE SISTEMA OPERACIONAL

Agora que já foi visto o tipo de aplicação do 80386, vamos ver como o 80386 suporta o sistema operacional que manipula o ambiente de aplicação. Desde que as requisi-

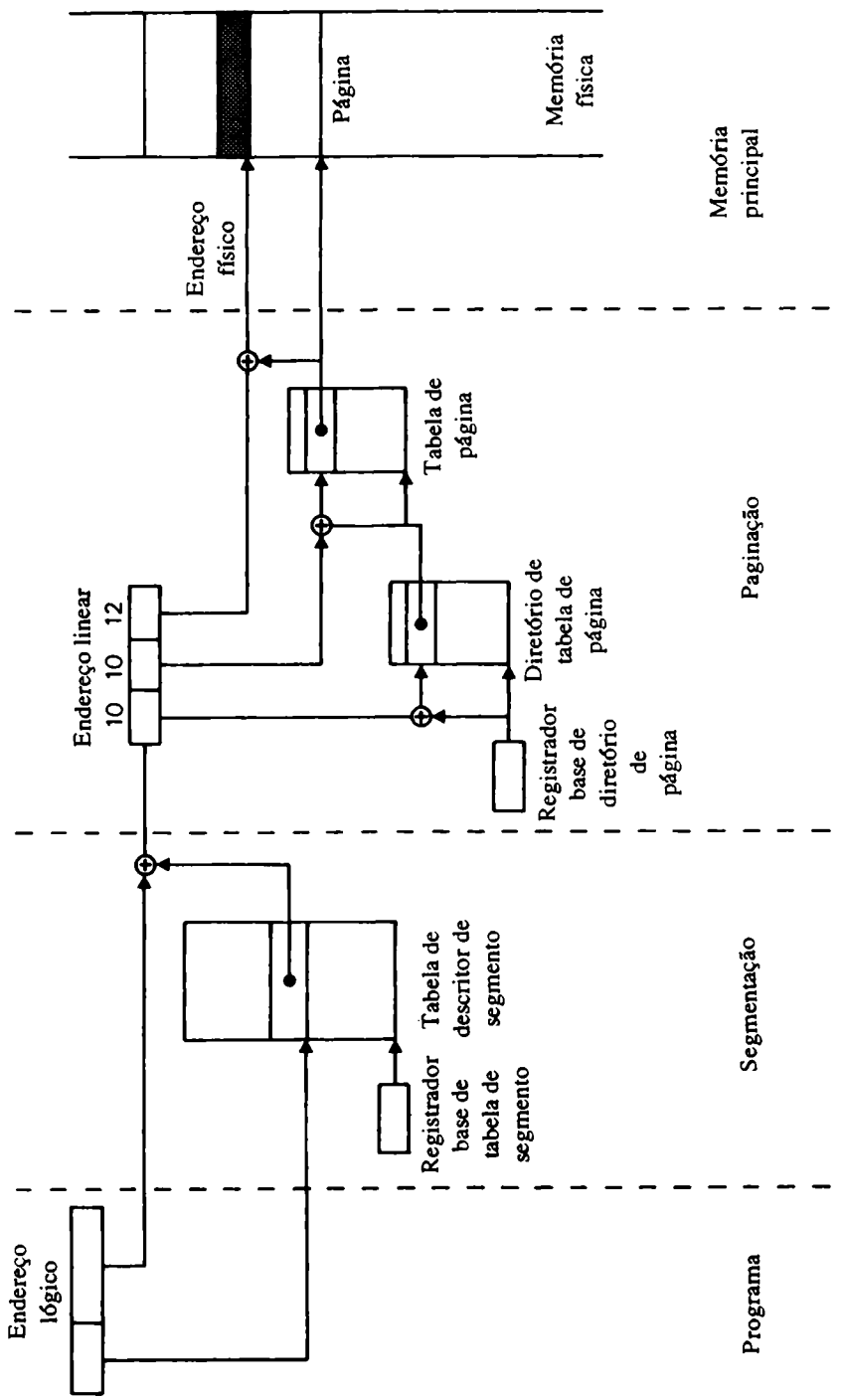


Figura 5.3 Tradução de endereço linear para físico.

ções de capacidade do sistema operacional estão se tornando mais e mais sofisticadas, o suporte a sistema operacional tem recebido grande atenção através do desenvolvimento do 80386. Incluindo-se várias características necessárias em um sistema operacional em pastilha avançado, o 80386 provê o trabalho com quadros para projetar um sistema operacional rápido, simples e padronizado. O suporte de operação do 80386 é totalmente transparente sob o ponto de vista da aplicação da máquina. As funções em pastilha providas pelo sistema operacional do 80386 são:

Dois níveis de proteção/gerenciamento de memória:

- sistema de segmentação com segmentos de 4 gigabytes
- sistema de paginação com 4 kbytes de páginas
- hardware voltado para isolamento e proteção de segmento/página/tarefa

Suporte a multitarefa:

- chaveamento de tarefa por hardware rápido
- execução múltipla e meio de endereçamento em uma base por tarefa/por segmento (transições dinâmicas).

5.3.1 GERENCIAMENTO DE MEMÓRIA

Sistemas operacionais multiusuário/multitarefa devem prover um meio para alocar eficientemente a memória do sistema para todas as aplicações. Em adição à alocação de memória, o SO deve prover um meio para isolar e proteger cada aplicação das outras. Ambas, paginação e segmentação, podem ser usadas para reforçar o isolamento. As seções seguintes descreverão características de relocação de endereços e proteção de memória em segmentação e paginação.

5.3.1.1 SEGMENTAÇÃO PROTEGIDA

O 80386 implementa um sistema de gerenciamento de memória segmentado baseado em descritor. Cada tarefa pode ter vários segmentos definidos (mais que 16384 ao mesmo tempo) para descrever sua visão de memória lógica. O descritor para cada segmento, como mostrado na Figura 5.4, define o segmento completamente. O descritor é armazenado na memória do sistema em uma das duas tabelas de descritores correntes disponíveis para cada tarefa; um descritor reside na tabela de descritores global do sistema (GDT), ou na tabela de descritores local de tarefa (LDT). Uma tabela particular e descritor dentro daquela tabela é selecionada pelo seletor de segmento (Figura 5.5).

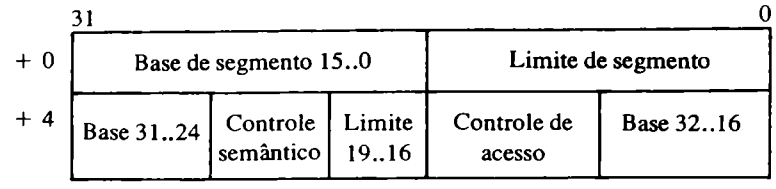
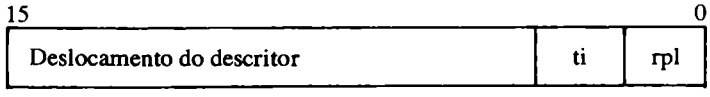


Figura 5.4 Formato do escritor de segmento.



ti = indicador de tabela (1 = LDT/0 = GDT).
rpl = nível de segurança do requisitor.

Figura 5.5 Formato do seletor de segmento.

O campo de *controle de acesso* do descritor provê tipos diferentes de segmentos a serem usados. Por exemplo, pode ser definido um segmento de dados só para leitura para armazenar dados de configuração de sistema sensíveis, ou um segmento de código somente para execução pode ser criado para assegurar que o programa não será violado. Outros tipos de sistemas são disponíveis para otimizar a velocidade de execução, tal como o tipo de segmento de estado de tarefa. O sistema operacional tem a liberdade de criar segmentos dinamicamente como requisitados pelas aplicações em execução. Em adição à relocação e proteção, os descritores provêem vários controles semânticos para cada segmento; essa capacidade é descrita futuramente. Além disso, a combinação de 16384 segmentos por tarefa com deslocamentos (offsets) de 32 bits do 80386 provê um intervalo de endereços lógicos de 64 terabytes para cada tarefa.

5.3.1.2 PROTEÇÃO SEGMENTADA

Em adição à tradução de endereços lógicos para linear, o mecanismo de segmentação do 80386 fornece a capacidade para incorporar vários graus de proteção no projeto do sistema. A integridade do endereço do programa está sempre presente na forma de base e campos de limites de um descritor. Um deslocamento gerado por aplicação causará um acesso ao endereço linear (base + deslocamento). Se o deslocamento for maior que o limite do segmento, uma violação de proteção ocorrerá. Um bit de controle de semântica, chamado *bit de granularidade* (G-bit), determina a interpretação do campo de limite dentro do descritor: um valor 0 indica um limite absoluto (limite granular do byte), enquanto 1 indica que o limite do segmento é o campo de limite * 4 k (limite granular de página).

Outra capacidade de proteção do 80386 é a proteção de nível de programa baseado em níveis privilegiados. O controle de acesso de cada descritor de segmento contém *um nível de privilégio de descritor* (DPL), que identifica o privilégio do segmento. Todos os privilégios estão entre 0 e 3, com 0 sendo o mais privilegiado. Em qualquer ponto no tempo, o 80386 está operando em *algum nível de privilégio corrente* (CPL), que é determinado pelo nível de privilégio do segmento de código sendo executado.

A Figura 5.6 ilustra o particionamento do sistema em níveis de privilégio. Como regra, quando executando em um dado nível de privilégio, um programa pode acessar segmentos que não são mais privilegiados que ele. Em adição, um programa pode chamar outros programas que são no mínimo tão privilegiados quanto ele. Transferências internníveis são providas por um descritor especial chamado *call gate*, que provê um ponto de entrada controlado para rotinas mais privilegiadas.

A Figura 5.6 também mostra a capacidade de proteção intertarefa do 80386. Essa capacidade existe a partir da opção de definir uma tabela descritora local separada (LDT) para cada tarefa. Usar uma tabela separada, LDT, resulta em um intervalo de endereço isolado logicamente para cada tarefa.

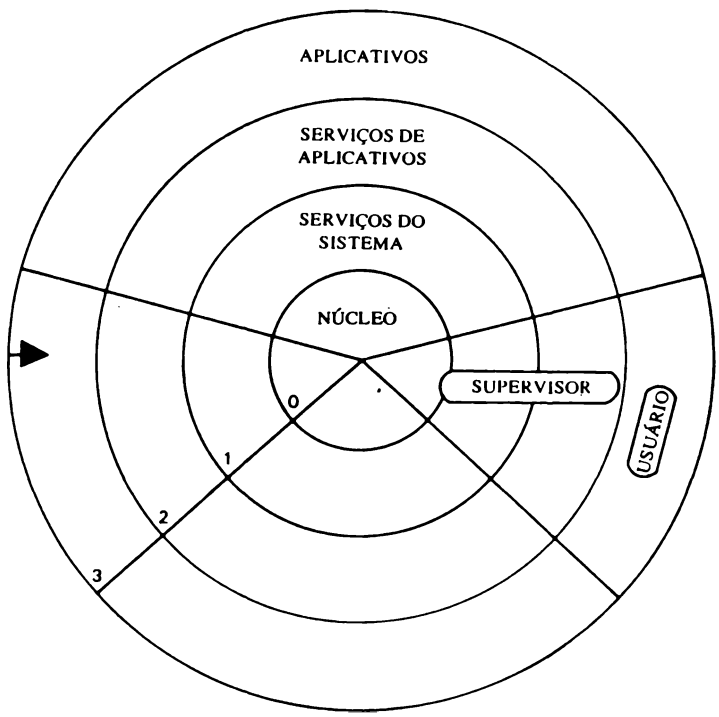


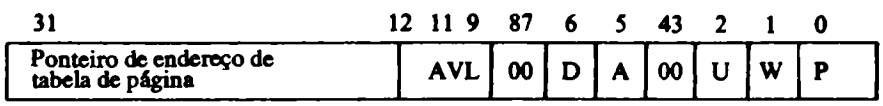
Figura 5.6 Níveis de privilégio.

5.3.1.3 CONTROLES SEMÂNTICOS

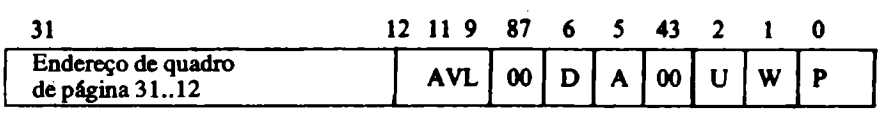
Vários outros bits de controle são de interesse. O bit de *tamanho default* (D-bit-size bit) para código de segmentos determina se devem ser assumidas operações de 32 (D = 1) ou 16 bits (D = 0). O bit de *expansão* (E-bit – expand down bit) para segmentos de dados define se um segmento de dado é uma expansão de pilha para baixo ou para cima (o campo de limite é também interproteção apropriada). Note que todos esses controles semânticos são alteráveis de um segmento para o próximo, portanto é possível chavear dinamicamente entre códigos de 16 e 32 bits. Dessa maneira, o processador adapta-se ao modo de processamento que foi selecionado para cada segmento e tarefa, permitindo chaveamentos fáceis entre códigos não alterados do 8086 ou 80286 e o novo código de 32 bits.

5.3.1.4 GERENCIAMENTO DE MEMÓRIA PAGINADA

O 80386 incorpora um mecanismo de tradução de página de 2 níveis que permite o mapeamento de qualquer página de endereços de memória linear de 4 k em qualquer quadro de memória física de 4 k. O mapa de página de 2 níveis para a tarefa corrente é alocado na memória pelo ponteiro no registrador de controle 3 (CR3), ou no *registrador base de diretório de página*. Todas as tabelas são alocadas em limites de página para manipulação eficiente de tabelas. Para cada página no intervalo de memória lógica, um descritor de página é definido (veja Figura 5.7). O descritor de página contém o endereço físico correspondente a cada página lógica, e também outras informações de estado de página.



Descritor de diretório de página



Descritor de tabela de página

AVL = Disponível.
D = Sujo.
A = Acessado.
U = Usuário (1)/ supervisor (0).
W = De escrita (1)/ de leitura apenas (0).
P = Presente.

Figura 5.7 Descritores de tabela de página e diretório.

Para aumentar a velocidade de tradução de endereço linear para físico, o 80386 contém um cache de descritores de página, conhecido como *vetor de saída de tradução* (TLB – *Translation Lookaside Buffer*). O TLB de 32 entradas resulta em uma percentagem maior que 98% de acerto, o que significa que, em mais de 98% do tempo, a tradução não tem efeito no desempenho. O TLB do 386 é ilustrado na Figura 5.8.

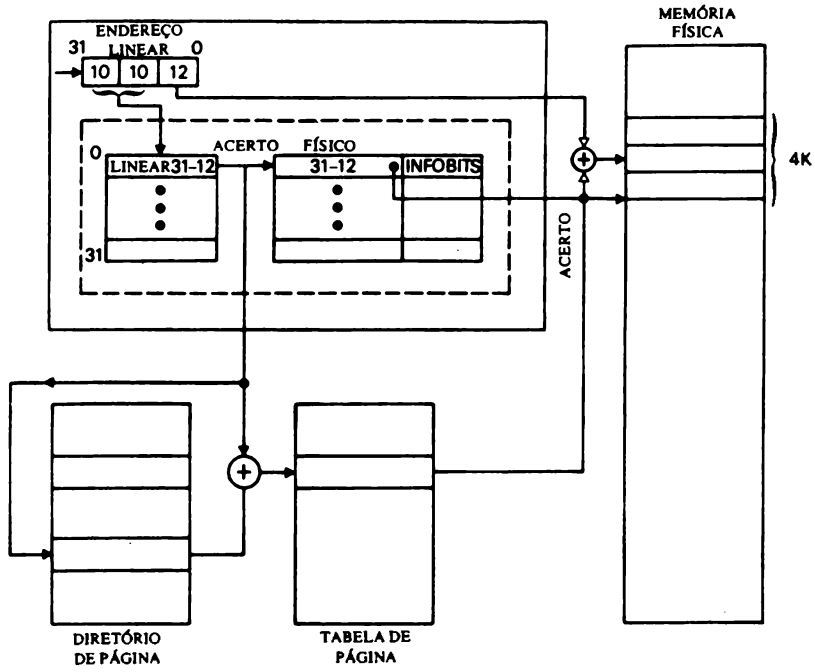


Figura 5.8 Buffer de antecipação de tradução.

A Tabela 5.2 mostra os diferentes graus de proteção que podem ser atribuídos a cada página, o que é provido pelo bit do usuário: o mapa de níveis de proteção de página para os níveis de proteção de segmentação, com o mapeamento de nível do usuário para um CPL de 3 e o mapa de nível de supervisor para as CPL de 0 a 2. Também são disponíveis dois bits de estado para seguir passo a passo a página em uso. O bit *acessado* é verdadeiro se a página ou o diretório de página foi acessado pela MMU. O bit de *sujo* é verdadeiro se a página ou o diretório foi escrito.

O suporte à paginação do 386 provê uma base poderosa para implementação de serviços essenciais de sistema operacional necessários para suportar gerenciamento de memória virtual paginada: alocação de página de memória, substituição de página de memória e armazenamento de página virtual na memória secundária.

Tabela 5.2 Página de proteção usuário/supervisor

Descritor de diretório de página	*Descritor de quadro de página*	*Acesso*			
U/S	*W/R*	*U/S*	*W/R*	*Usuário*	*Supervisor*
0	0	0	0	Nada.	Leitura/escrita
0	0	0	1	Nada	Leitura/escrita
0	0	1	0	Nada	Leitura/escrita
0	0	1	1	Nada	Leitura/escrita
0	1	0	0	Nada	Leitura/escrita
0	1	0	1	Nada	Leitura/escrita
0	1	1	0	Nada	Leitura/escrita
0	1	1	1	Nada	Leitura/escrita
1	0	0	0	Nada	Leitura/escrita
1	0	0	1	Nada	Leitura/escrita
1	0	1	0	Apenas leitura	Leitura/escrita
1	0	1	1	Apenas leitura	Leitura/escrita
1	1	0	0	Nada	Leitura/escrita
1	1	0	1	Nada	Leitura/escrita
1	1	1	0	Apenas leitura	Leitura/escrita
1	1	1	1	Leitura/escrita	Leitura/escrita

5.3.1.5 USO DO GERENCIAMENTO DE MEMÓRIA DO 80386

A flexibilidade da MMU do 80386, descrita anteriormente em detalhes, dá ao usuário ferramentas para projetar arquiteturas de intervalo de endereços diretamente compatíveis com diversas aplicações. A manipulação dos mecanismos de segmentação e paginação permite serem implementados diretamente os quatro intervalos de endereços maiores:

- modelo linear (efetivamente não paginação ou segmentação)
- modelo linear paginado (relocação de página)
- modelo segmentado e paginado (usa todos os recursos do 80386)
- modelo segmentado (relocação de página)

Por exemplo, o modelo de segmentado e paginado é o modelo usado no Sistema Unix System V, onde cada tarefa tem uma unidade de segmento de código, segmento de dados e segmento de pilha; a unidade de segmentação do 80386 provê segmentos por ta-

refa, enquanto a unidade de paginação permite gerenciamento efetivo do conjunto de trabalho a ser mantido na memória principal.

O modelo segmentado é muito útil quando os tamanhos de segmentos não são grandes e quando a compatibilidade com a família de processadores iAPX 86 é desejada. Adicionalmente, o esquema de proteção provido é muito importante quando a segurança do sistema e/ou a integridade são benefícios desejados.

O modelo linear permite serem implementados sistemas operacionais bem compactos, já que segmentação e paginação podem ser configuradas fora do sistema; nesse modelo, o endereço do programa é o endereço físico. Isto é tipicamente o projeto preferido em muitas aplicações de tempo real.

Finalmente, alguns projetistas preferem uma arquitetura lógica com a paginação usada para gerenciar a memória física. O 80386 provê essa visão de memória no modelo linear paginado e, desde que o mecanismo de paginação é em pastilha, o desempenho do 80386 é superior a arquiteturas com MMU externa usando esse modelo.

5.3.2 MULTITAREFAS E AMBIENTES MÚLTIPLOS

Lembre-se de que um dos tipos de segmentos especiais providos no 80386 é o *segmento de estado de tarefa*, ou TSS. Esse tipo de segmento armazena toda informação de estado de uma tarefa, auxiliando o hardware para ser usado para chavear tarefas de maneira extremamente rápida, armazenando e carregando informações de novas e velhas TSS em uma instrução. O 80386 mantém compatibilidade com o mecanismo de chaveamento de tarefas provendo suporte aos formatos de TSS 80286 e 80386 – os formatos 80386 e 80286 são diferentes, já que o 80286 é um processador de 16 bits. Adicionalmente, o TSS do 80386 inclui o estado do registrador de base de diretório de página, permitindo um mapeamento de página diferente para ser usado em cada tarefa.

5.3.3 EXECUÇÃO 8086 VIRTUAL

O 80386 inclui um modelo para emular a operação do processador 8086/8088 dentro de seu modelo virtual protegido. Essa capacidade é controlada pelo bit de *modelo virtual* (VM-bit) no *registrador de estado*. O atributo mais significante desse modelo é a carga e o uso de registradores de segmento. Em um segmento de código 8086 virtual, as semânticas de registrador de segmento são as mesmas do 8086. Assim, códigos não alterados do 8086 (mesmo códigos que incluem práticas não recomendadas do uso de registra-

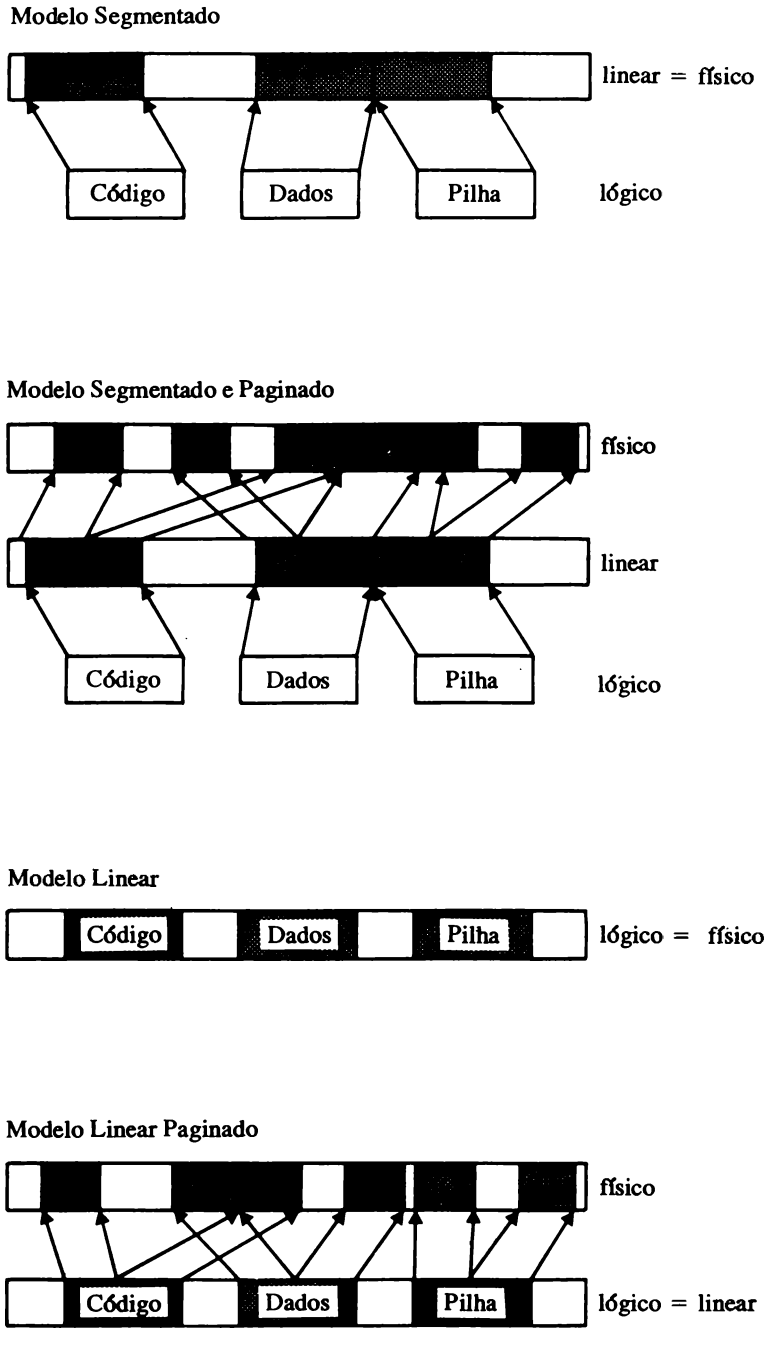


Figura 5.9 Espaços de endereçamento do 80386.

dores de segmentos para armazenamento temporário) podem ser executados no modelo protegido do 80386. Isto torna possível a fácil migração de aproximadamente quatro bilhões de dólares em software existentes atualmente para o 8086/8088.

Quando no modo virtual 86, o 80386 parece-se com o 8086 para o programa de aplicação – o valor do registrador de seletor de segmento é deslocado para a esquerda em 4 bits para computar a base do segmento real. Como nos modelos atuais do 8086, e no 80286 e 80386, o tamanho de cada segmento é 64 k. Ainda que o 80386 seja capaz de gerar deslocamentos (offset) maiores que 64 k, essa memória não é acessível. As interrupções causam um chaveamento automático fora do modelo 86 virtual, onde o manipulador de interrupção do sistema operacional pode tratar a interrupção, fora do programa 8086, se for apropriado.

Cada seção de código virtual 8086 gerará endereços lineares em um campo de endereços de 0-1 megabytes. Para prover intervalos de endereços separados para cada um desses, um diretório de página deve ser usado para cada tarefa. O mecanismo de paginação pode ser usado também para simular os endereços envolvidos nos 1 megabyte que estavam no 8086. E o esquema de proteção é ainda mantido e bem em modelo 86 virtual como programas em modelos 86 virtual sempre executam no menor nível de privilégio, nível 3.

5.3.4 AMBIENTES DE MULTITAREFAS

Um dos aspectos mais poderosos do 80386 é percebido quando os mecanismos de controle de semântica, arquitetura de intervalo de endereços flexível e multitarefa são combinados. Por exemplo, aplicações escritas para cada um dos ambientes seguintes poderiam existir como tarefas concorrentes no 386 – de fato, poderiam existir exemplos múltiplos para cada ambiente executando concorrentemente:

- modelo segmentado e paginado executando código do Sistema Unix System V de 32 bits e dados com 64 terabytes de endereços disponíveis;
- modelo segmentado e paginado executando código objeto do 80286 de 16 bits de produtos anteriores baseado no 286 – a paginação provê gerenciamento de memória física e é transparente para as aplicações;
- modelo segmentado e paginado executando código objeto do 8088, tal como programas do PC, em modo 8086 virtual de 16 bits;
- modelo linear paginado executando código de 32 bits de ambientes Unix linear, tal como um Unix 42 , da Berkeley.

Desde que todos os bits de controles semânticos do 386 são chaveados quando um chaveamento de tarefa é executado, um novo ambiente pode ser integrado com uma instrução. Com esse mecanismo, o projetista de sistema tem um alto grau de flexibilidade e poder para projetar um sistema ótimo para sua aplicação.

Nas seções restantes deste capítulo serão detalhados os subsistemas de hardware do 80386, permitindo ao projetista de sistema utilizar o poder do 80386.

5.4 BARRAMENTO EXTERNO DO 80386

O barramento externo do 80386 tem sido otimizado para permitir acesso eficiente à memória externa e E/S. Somente dois ciclos de relógio são necessários para que o processador execute um acesso completo ao barramento, permitindo, portanto, manter o barramento acima de 32 Mbytes por segundo a 16 MHz. Essa velocidade rápida de barramento é ideal para o uso com o cache de memória para obter a banda máxima de barramento com componentes de custo baixo.

O 80386 tem um barramento de dados de 32 bits e um barramento de endereços físico de 32 bits. O conjunto de instruções pode acessar dados no formato em byte, palavra ou palavra dupla. Se um acesso à palavra ou à palavra dupla não é alinhado e cruza duas palavras duplas no barramento externo, o 80386 gera automaticamente dois ciclos de barramento para executar o acesso.

O 80386 também acopla memórias vagarosas e subsistemas periféricos. Uma entrada READY no processador é usada para indicar que o acesso ao barramento foi completado. O subsistema externo pode manter o READY inativo para fazer o processador 80386 estender o ciclo de barramento até que o subsistema tenha completado o acesso.

A maioria dos sistemas de memória de 32 bits, independentemente de ser processador individual, requer apenas os bits mais significativos do endereço e usa habilitadores individuais para os quatro bytes menos significativos, ao contrário dos dois bits menos significativos do endereço, A1 e A0. O 80386, portanto, gera diretamente os bits mais significativos do endereço (A31:2) e os bits habilitadores individuais, apesar dos bits menos significativos do endereço, para interfacear de forma simples e eficiente a memória. O método alternativo de gerar todas as linhas de endereços com sinais para indicar o tamanho do acesso (byte, palavra ou palavra dupla) requer um número idêntico de pinos no processador e torna a lógica de interface externa mais vagarosa e significativamente mais complexa. Se, entretanto, os dois bits menos significativos do endereço devem ser gerados, eles podem facilmente ser decodificados a partir dos bytes habilitadores, usando qua-

tro portas. Geralmente, o único momento em que os bits menos significativos de endereços são necessários é quando um barramento-padrão do sistema é usado.

Uma das considerações mais significativas no projeto do 80386 foi a acomodação de uma grande variedade de periféricos e tipos de memória. Para simplificar a lógica de interface de barramento, somente um sinal é usado para indicar o início do ciclo de barramento (Address Status – ADS), então sinais individuais são usados para indicar escrita/leitura (W/R), E/S de memória (M/IO) e dados/código (D/C). Adicionalmente, o sinal $\overline{\text{LOCK}}$ indica que uma operação de barramento crítica está sendo executada para outros mestres de barramento, que, conseqüentemente, não são autorizados a obter acesso ao barramento enquanto aquela operação crítica estiver em execução. Ele pode ser usado em sistemas de multiprocessamento para implementar semáforos teste-e-assinalamento confiável ou para acessar outros dados críticos. O $\overline{\text{LOCK}}$ automaticamente ativado pelo 80386 para a instrução de troca (XCHG), durante a atualização das tabelas de descritores e páginas e durante os ciclos de reconhecimento de interrupção. (O programador pode, naturalmente, ativar o sinal $\overline{\text{LOCK}}$ explicitamente durante certas instruções.)

Os sinais de barramento do 80386 não somente são escolhidos cuidadosamente para fazer interfaceamento para a memória e E/S tão simples quanto possível, mas todo o barramento é projetado para permitir um desempenho bem alto com custo de memória baixo. Como outros processadores, o 80386 usa barramento de dados e endereços separados (cada um usa um conjunto separado de pinos em vez de multiplexá-los nos mesmos pinos, o que permite que o barramento execute mais rapidamente). Mas o 80386 dá um passo à frente por também canalizar externamente o barramento de dados e endereços. Se desejado, o endereço para o próximo ciclo de barramento pode ser canalizado. O próximo endereço pode ser escrito na saída antes que seja completado o ciclo de barramento corrente. Isto permite a todos os atrasos de saída de endereços e buffers sobrepor completamente o ciclo de barramento corrente e provê mais tempo de acesso para os sistemas de memória e E/S, sem nenhum efeito no desempenho. Portanto, um alto desempenho pode ser obtido em componentes de E/S e memórias menos caras.

Para uma flexibilidade máxima, um hardware externo tem um controle completo de quando e se o próximo ciclo de barramento será canalizado. Se o endereço não é canalizado (Figura 5.10), ele se mantém válido até o fim do ciclo do barramento. Quando o hardware externo ativa o próximo pino de endereço do 80386 (NA), o barramento de endereço é canalizado (Figura 5.11), e o endereço para o próximo ciclo de barramento torna-se disponível antes do fim do ciclo de barramento corrente.

5.4.1 SUBSISTEMA DE CACHE E SRAM

Embora a memória DRAM possa ser usada com um desempenho bem alto, bandas de barramento alto podem ser obtidas usando-se RAM estáticas (SRAM). SRAM a 16 MHz, 55 ns são rápidas o suficiente para completar um acesso dentro de dois ciclos de relógio sem interfoliar e sem canalização de endereços. Isto significa que todos os acessos de barramento podem ser completados com zero estado de espera. O 80386 acomoda otimamente essas memórias muito rápidas sem forçar o endereço a ser canalizado.

Quando não se conhece o endereço do próximo ciclo de barramento até depois do fim do ciclo de barramento corrente (como saltos indiretos ou acessos de dados dependentes) ou seguido de um ciclo de barramento sem função (idle), se o 80386 forçou o endereço a ser canalizado um ciclo de relógio antes do início do próximo ciclo de barramento, o acesso é realmente atrasado de um ciclo de relógio. Em vez disso, o 80386 pode gerar ciclos de barramento com ou sem canalização de endereço. A permissão de canalização é mais otimizada para memórias mais lentas, como DRAM, enquanto não forçar a canalização é mais otimizado para memórias rápidas, tais como SRAM. Diferentemente de outros microprocessadores, o 80386 é suficientemente flexível para usar uma interface simples de memória que provê todos os benefícios de barramentos de endereço tanto canalizados como não canalizados.

Portanto, o 80386 não canaliza o endereço quando este não está disponível suficientemente cedo, e se o subsistema de memória pode completar o acesso em dois relógios, ele simplesmente ativa o sinal $\overline{\text{READY}}$, de forma que não há punição pela não canalização do endereço.

Embora os SRAM consigam o mais alto desempenho de barramento, DRAM provêem muito mais espaço de memória por um custo e espaço de placa muito menores.

Um sistema de cache pode ser usado para tomar vantagem das melhores características de ambos. Um sistema de cache pode ser construído para permitir que o código e os dados mais freqüentemente acessados sejam armazenados em SRAM e colocar o código e os dados menos freqüentemente acessados em DRAM mais lentas. Isso diminui o custo geral da memória e ainda provê um desempenho com quase zero estado de espera.

A Figura 5.12 mostra um subsistema de cache que pode completar ciclos de barramento com zero estados de espera com ou sem canalização de endereço. Para um sistema de 12 Hz ele usa RAM de dados de 95 ns e RAM de sinalização de 75 ns. Para um sistema de 16 MHz, usa correspondentemente RAM de 55 e 35 ns. Para manter o controle

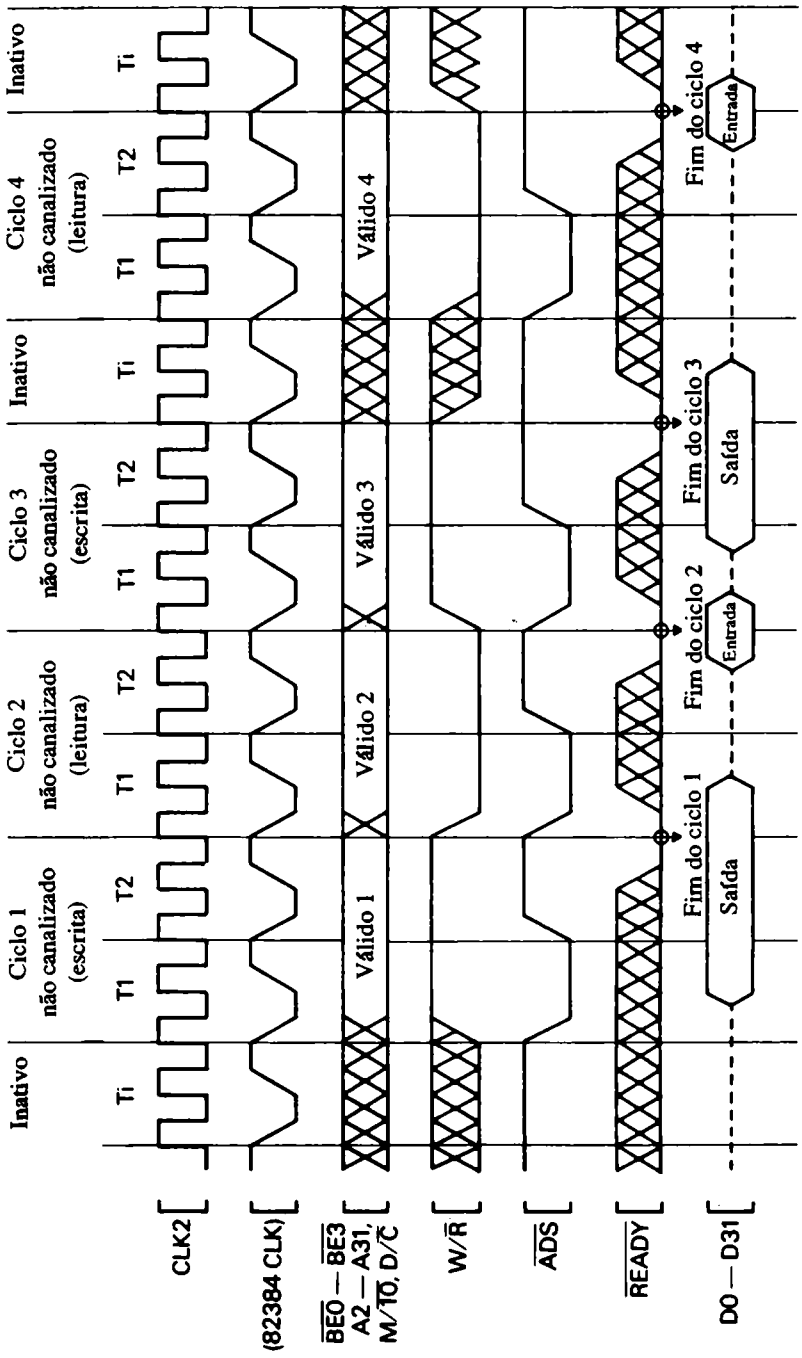


Figura 5.10 Ciclos de barramento com endereços não canalizados (zero ciclo de espera).

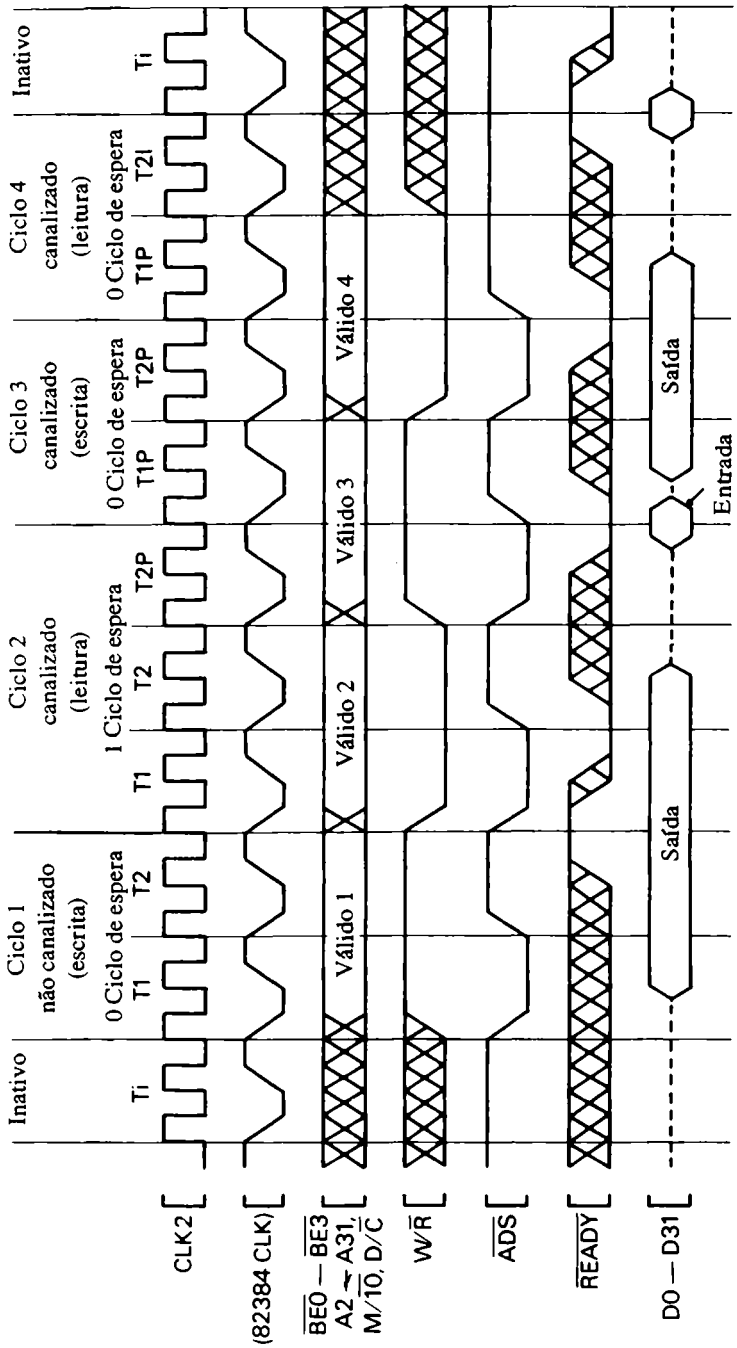


Figura 5.11 Ciclos de barramento do 80386 com e sem canalização e ciclos de espera.

do cache simples e efetivo em custo, um cache direto com bloco de 4 bytes e escrita através do mesmo é usado.

Com 32 kbytes de cache, esse cache possuirá uma taxa de acerto aproximada de 86%, o que significa que, em 86% do tempo, dados ou código estão no cache e os acessos de barramento requerem zero estado de espera. Com 64 kbytes de cache, essa taxa cresce para 92%.

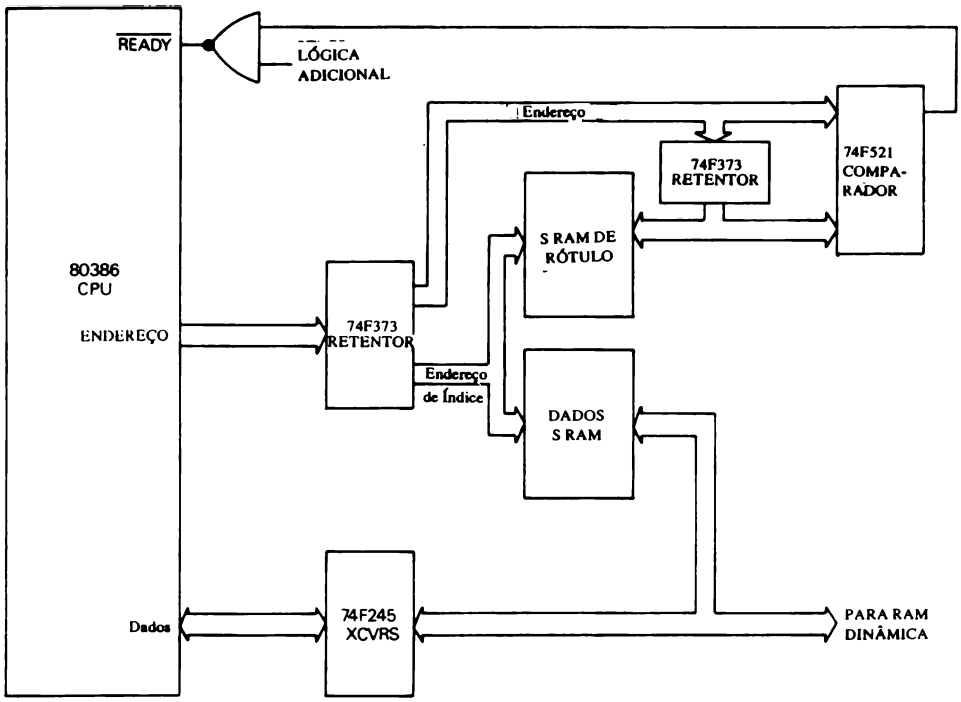


Figura 5.12 Subsistema de cache.

5.4.1.1 EFETIVIDADE DO CACHE

Em muitos casos, projetos de alto desempenho baseados em processador requerem uma vasta quantidade de memória de alta velocidade, de forma que o processador possa rodar programas grandes ou acessar rapidamente uma vasta quantidade de dados. O projetista de um sistema de alto desempenho tem o desafio de prover essa grande quantidade de memória de alta velocidade a um custo razoável.

Para o máximo desempenho, uma grande aplicação num microprocessador de

alta velocidade, tal como o 80386 de 16 MHz, pode requerer diversos megabytes de memória com zero estado de espera (requerendo que o tempo total de acesso à memória, incluindo atrasos em buffers, não exceda 125 ns). Enquanto SRAM podem prover a velocidade necessária, elas se tornam proibitivamente caras para a maioria das aplicações quando uma grande quantidade de memória é necessária (muitas das aplicações atuais não podem suportar o custo de mais de 64 K de SRAM). Do outro lado do espectro, DRAM provêem grandes quantidades de memória de baixo custo, mas as DRAM de baixo custo de hoje em dia são muito lentas para permitir ao processador rodar sem estados de espera (uma aplicação típica pode hoje possuir muitos megabytes de DRAM de 150 ns).

A solução mais efetiva a nível de custo na maior parte dos casos é projetar uma memória cache. Uma memória cache é simplesmente uma memória de alta velocidade (tipicamente SRAM) que é colocada entre o processador e uma memória maior e mais lenta (tipicamente DRAM). A maioria dos programas possui o hábito de acessar novamente as mesmas posições de memória logo após a primeira referência a essa posição. Para a primeira referência, o processador tem geralmente de esperar o acesso da memória mais lenta. Enquanto o processador está fazendo esse acesso, o cache também coloca o dado em sua memória. Então, se o processador necessitar novamente do dado, bastará acessá-lo no cache de alta velocidade. Pelo fato de o cache ser menor que a memória mais lenta, ele não pode armazenar todos os dados contidos nesta. Portanto, quando o cache está cheio e necessita armazenar uma nova posição de memória, ele precisa trocar o conteúdo de uma posição antiga com a nova. O objetivo de um cache é manter as posições que o processador irá mais provavelmente precisar, de forma que os acessos a essas posições ocorram em memória rápida ao invés de DRAM mais lentas.

A maioria dos caches mantém a informação mais recentemente usada (código e dados) disponível em memória de alta velocidade, porque a maioria dos programas apresenta localidade de referência; isto significa que é alta a probabilidade de essa informação ser acessada em um futuro próximo. A percentagem de acessos do processador que encontra a informação já referenciada no cache é chamada *taxa de acerto* (hit rate).

A taxa de acerto do cache é um indicador muito importante de sua efetividade. Quanto mais alta a taxa de acerto, mais freqüentemente os dados ou código são encontrados no cache, e um acesso de cache rápido é usado, em vez do acesso menos rápido da memória. Há vários fatores que determinam a taxa de acerto do cache, tais como a quantidade de memória do cache e o método de determinar quais endereços podem ser substituídos quando o cache está completo, mas a taxa de acerto indica a efetividade do cache global.

Em quase todos os sistemas de cache de microprocessadores, o acesso de baixa

velocidade da DRAM não é iniciado antes que seja determinado se o dado ou código foi encontrado no cache, o que usa normalmente um ou dois relógios depois que o ciclo de barramento inicia. Portanto, quando o dado ou código não é encontrado no cache e o ciclo da DRAM é necessário, o ciclo da DRAM realmente demora mais do que se o cache não tivesse sido usado, porque ele começa mais tarde. Se a taxa de acerto for baixa o suficiente, significando que a falta de cache prevalece, então o uso do cache pode realmente diminuir o desempenho global. Um cache com taxa de acerto de 50-60% é de um valor muito questionável. Caches com menos de 4 k geralmente não são úteis.

Não somente uma baixa taxa de acerto indica que o cache é relativamente inefetivo, como um cache com uma taxa muito baixa de acerto pode realmente degradar o desempenho. O caminho de custo mais baixo para incrementar a taxa de acerto é aumentar o tamanho do cache. Portanto, em vez de incluir-se um cache em pastilha limitado, o 80386 suporta um cache de dois ciclos de relógios externo, no qual o tamanho e a taxa de acerto, e assim a efetividade global, podem ser tão grandes quanto desejados.

O barramento rápido do 80386 permite que referências à memória externa sejam feitas no mesmo número de ciclos de relógios que outros microprocessadores requerem para acessar caches em pastilha. Enquanto a tecnologia corrente de silício prevê caches de propósito geral para serem alocados na pastilha do processador, caches de MMU em pastilha relativamente pequenos podem ser muito efetivos. Os 80386 provisionam os descritores de segmentação e paginação. Alocando o cache de descritor da página da TLB (vetor de saída de tradução) de 32 entradas no 80386, uma percentagem de mais de 98% pode ser alcançada.

5.5 PERIFÉRICOS

A estrutura de barramento do 80386 foi simplificada, minimizando o custo de conexão lógica para a interface de periféricos comuns e ROM/EPROM. Uma lógica discreta, a PALS ou vetores de porta podem ser configurados eficientemente, interfaces usuais para conjuntos específicos de periféricos necessários.

O 80386 tem uma entrada chamada Bus Size 16 Bit (BS16) que simplifica o uso do processador com o barramento de 32 ou 16 bits. Isto permite controle do tamanho do barramento externo para o ciclo de barramento corrente. Quando ativo, somente os 16 pinos de dados mais baixos são usados (D15:0), e acessos não alinhados ou de 32 bits são quebrados em múltiplos acessos de palavras de 16 bits. Quando o BS16 está inativo, o 80386 usa todo o seu barramento de dados de 32 bits (D31:0) para acessos de bytes, palavras ou palavras duplas. (Em virtude do 80386 ter instruções que executam acessos a byte, palavras e palavras duplas, periféricos de 8, 16 e 32 bits podem ser conectados diretamente ao barramento local.)

Essa capacidade de barramento de 16 bits reduz o custo do sistema e consome menos interconexões de barramento, dando uma vantagem adicional na alocação de periféricos na grandeza de palavras de 16 bits para duplicar os endereços de E/S usados em outros sistemas de 16 bits oferecendo compatibilidade de software com os sistemas 8086 e 80286.

Um ciclo de barramento que usa somente os dois bytes de barramento de dados menos significativos (ciclo com BE2 e BE3 inativos) sempre usa somente D15:0 e, portanto, BS16 é ignorado. Um ciclo de barramento Read com BE0 e BE1 inativos poderia normalmente usar D31:16 para ler os dados, mas se o BS16 está inativo o dado é lido de D15:0. Um ciclo de escrita com BE0 e BE1 inativos sempre duplicarão o dado de escrita em D31:16 em D15:0, portanto todos os 16 bits podem ser utilizados pelo hardware e BS16 é novamente ignorado pelo 80386. Todos os acessos restantes são automaticamente convertidos pelo 80386 em dois ciclos de barramento. Essa facilidade executa ambos acessos de 16 e 32 bits sem gastar ciclos.

5.6 ACESSO DIRETO À MEMÓRIA

A movimentação de dados entre RAM e periféricos é possível que interrompa o processador, tendo-se salvo seu estado, executa-se a transferência e então restaura-se o estado do processador. Entretanto, é muito mais eficiente mover grandes quantidades de dados seqüenciais com um controlador otimizado para a tarefa, um *controlador de acesso direto à memória* (DMA), para não perturbar a execução do processador. O 80386 provê pinos HOLD e HLDA (reconhecimento de hold) que permitem um controlador de DMA ou outro processador que coloca os sinais do 80386 em alta impedância e controla o barramento para as transferências.

Um uso muito comum do DMA é executar transferências entre a memória principal e a memória de massa, tal como troca de memória virtual.

O 80386 permite segmentos de vários tipos, de um byte até 2^{32} (4 gigabytes), o que é suficiente para quase qualquer código ou estrutura de dados. Com a possibilidade de tal comprimento de segmento, é muito fácil que o programador aloque seus dados ou código em segmentos, mas poderia ser freqüentemente difícil, se não impossível, para o sistema operacional, implementar memória virtual para encontrar espaço em RAM física para trocar todo o segmento. Portanto o 80386 também implementa segmentação sob paginação. Se ela for usada, um segmento pode ser fatiado em páginas de tamanho regular, possivelmente com cada página correspondendo ao tamanho do bloco da memória de massa. Isto não só permite porções de segmentos residirem em RAM, como também per-

mite que os segmentos sejam partidos em várias partes para preencher qualquer RAM livre, mas não necessariamente contínua, eliminando problemas de fragmentação.

Enquanto a segmentação permitir ao programador de software alocar seu código e dados em estruturas naturais e eficientes e prover o sistema operacional com informação válida em que as porções de memória são boas candidatas para trocas, paginação dentro de segmentação permite eficiente implementação de hardware da memória virtual.

Na implementação de sistemas de paginação por demanda, um controlador DMA padrão pode ser usado para mover um ou mais blocos seqüenciais (páginas) de dados entre a RAM e a memória de massa. Depois de cada série de blocos seqüenciais ser movida, a controladora DMA normalmente interrompe o processador e espera por mais comandos do processador.

Se vários blocos estão sendo movidos em várias páginas não seqüenciais, o DMA deve parar após cada página e interromper o processador. O processador deve então enviar novos comandos para o controlador DMA para a próxima página. Isto diminuirá o desempenho do processador, porque ele deve parar e manipular uma interrupção depois de cada página. Também, toda a transferência de dados da memória de massa pode ser diminuída. A memória de massa é usualmente implementada como um disco que passa blocos seqüenciais pela cabeça de leitura/escrita. Se não há tempo suficiente entre o fim de uma página e o começo da próxima para o DMA interromper o processador e o processador para reprogramar o DMA, então a próxima página será perdida e o DMA terá de esperar que o disco seja lido novamente para continuar.

A pastilha controladora DMA Advanced Intel 82258 pode ser usada para resolver ambos os problemas: menos interrupções do processador e transferências de multipáginas mais rápidas. O processador pode gerar uma lista de blocos destinos/fontes para o DMA, por exemplo, trocar uma série de páginas não contínuas. Então o 82258 pode usar suas características de dados correntes automaticamente para trocar todas as páginas, sem interromper o processador e sem reprogramar o 82258 entre cada página não contínua. Portanto o 82258 provê desempenho excelente em meios de paginação por demanda.

5.7 EXEMPLO DE PROJETO: ESTAÇÃO DE TRABALHO DE ENGENHARIA

Uma engenharia de estação de trabalho tipicamente requer poder de computação alto e um espaço de memória grande para executar tarefas como simulação de projeto e verificação. Em adição, a habilidade para executar múltiplos programas permite que o projetista maximize sua efetividade porque ele pode trabalhar em uma parte do projeto

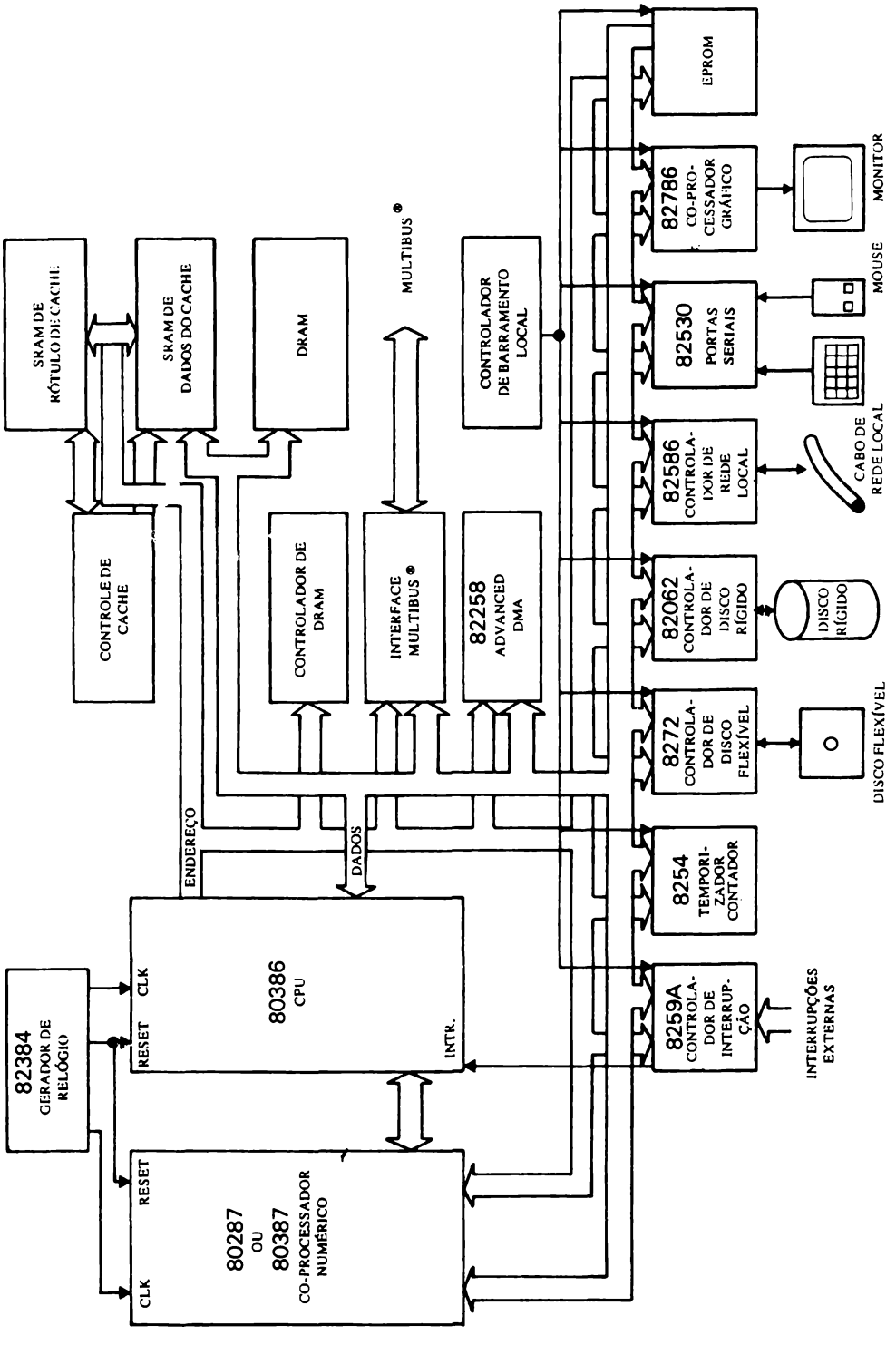


Figura 5.13 Estação de trabalho de engenharia.

enquanto simula outra parte. Em vários casos, a habilidade da estação de trabalho em rapidamente criar e manipular detalhadamente figuras gráficas de projetos é essencial também para permitir que o projetista entre e perceba seu projeto eficientemente. Finalmente, em virtude de a maioria dos projetistas trabalhar em grupos ao invés de individualmente, é importante que suas estações de trabalho possam comunicar-se facilmente.

A Figura 5.13 ilustra um projeto de sistema completo do 80386 que supre essas necessidades. Sua alta velocidade de processamento e espaço de endereços físico e virtual grandes permitem que a simulação seja executada eficientemente. Para um alto desempenho com uma grande quantia de DRAM de custo efetivo, um cache é usado para permitir operações com estados de espera perto de zero. A capacidade de segmentação e paginação, junto com o DMA Advanced 82258 e um controlador de disco fixo inteligente (82062), combinam para formar uma implementação poderosa e eficiente de memória virtual. O gerenciamento de memória em pastilha não somente provê tradução eficiente para memória virtual como também oferece proteção e capacidade de multitarefa.

Enquanto a velocidade computacional e a flexibilidade do 80386 permitem executar todos os cálculos, gráficos e funções de comunicação para algumas estações de trabalho, um desempenho maior pode ser obtido quando co-processadores especializados são usados. Os processadores numéricos 80287 ou 80387 podem ser usados para prover aritmética de ponto flutuante rápida, permitindo que simulações complexas e acuradas sejam completadas rapidamente. O co-processador de Gráficos/Display 82786 pode rapidamente gerar e manipular textos complexos e imagens gráficas tanto quanto criar um ambiente de janelas eficiente. O 80386 simplesmente supre comandos de alto nível de gráficos, e o 82786 executa tarefas tediosas de funções de atualização de pontos de imagens e display. O co-processador de Rede Local (LAN) pode manipular comunicação entre estações de trabalho em alta velocidade: enviando e recebendo pacotes de dados múltiplos, gerando protocolos e executando verificação de erros sem a intervenção do microprocessador.

Em torno do sistema, o 80386 também conecta periféricos escravos, incluindo duas portas seriais para teclado e modem e mouse, e um controlador de disco floppy. Esse subsistema 80386 poderia ser construído em uma placa com interfaces múltiplas para poder ser instalado em um sistema industrial padrão e prover acessos para os processadores e dispositivos periféricos em outras placas MULTIBUS dentro do mesmo sistema.

CAPÍTULO 6

O MOTOROLA 68020

Cyndy Zoch
Motorola Inc., USA

6.1 INTRODUÇÃO

O MC68020 é o primeiro microprocessador de 32 bits baseado na família M68000. Ele é implementado com registradores de 32 bits e passagem de dados e endereços de 32 bits, um rico conjunto de instruções e modos de endereçamento versáteis.

O MC68020 tem um código objeto compatível com os outros membros da família M68000 (o MC68000, MC68008, MC68010 e o MC68012). Foram acrescentadas características de um cache de instruções em pastilha, linguagem de alto nível e uma interface de co-processador flexível. Em adição, o MC68020 suporta um mecanismo de dimensionamento de barramento dinâmico que permite determinar o tamanho de porto em uma base ciclo-por-ciclo.

Este capítulo irá primeiramente cobrir a tecnologia de processamento e a arquitetura do MC68020, seguido de uma discussão sobre a organização de dados. Depois serão examinados a capacidade de endereçamento, conjunto de instruções e sinais de hardware. Então, algumas das características especiais do MC68020 serão detalhadas, incluindo o mecanismo de dimensionamento do barramento dinâmico, o cache de instrução em pastilha e a interface de co-processador. As três seções seguintes cobrirão gerenciamento de memória, suporte a sistemas de desenvolvimento e suporte a software. Finalmente, o capítulo se encerra com uma discussão sobre desempenho.

6.2 PROCESSAMENTO

O MC68020 é um produto de 60 anos-homem de empenho em projeto. O

processador é fabricado usando-se um processo de Semicondutor de Óxido de Metal Complementar de Alta Densidade (High-Complementary Metal Oxide Semiconductor – HCMOS) que provê uma combinação importante de desempenho, alta velocidade e consumo reduzido. Mais especificamente, o MC68020 roda em uma freqüência de relógio de 16,67 MHz e tem uma dissipação máxima de 1,75 W. O processador consiste em 200.000 transistores reais em uma superfície de 0.375×0.350 polegadas e é encapsulado em *grid pin array* de 114 pinos.

6.3 ARQUITETURA

A Figura 6.1 mostra um diagrama de blocos funcional do MC68020. O processador pode ser dividido em três seções principais, o controlador de barramento, a micromáquina e a área mista. A maioria dessas seções opera de modo completamente autônomo.

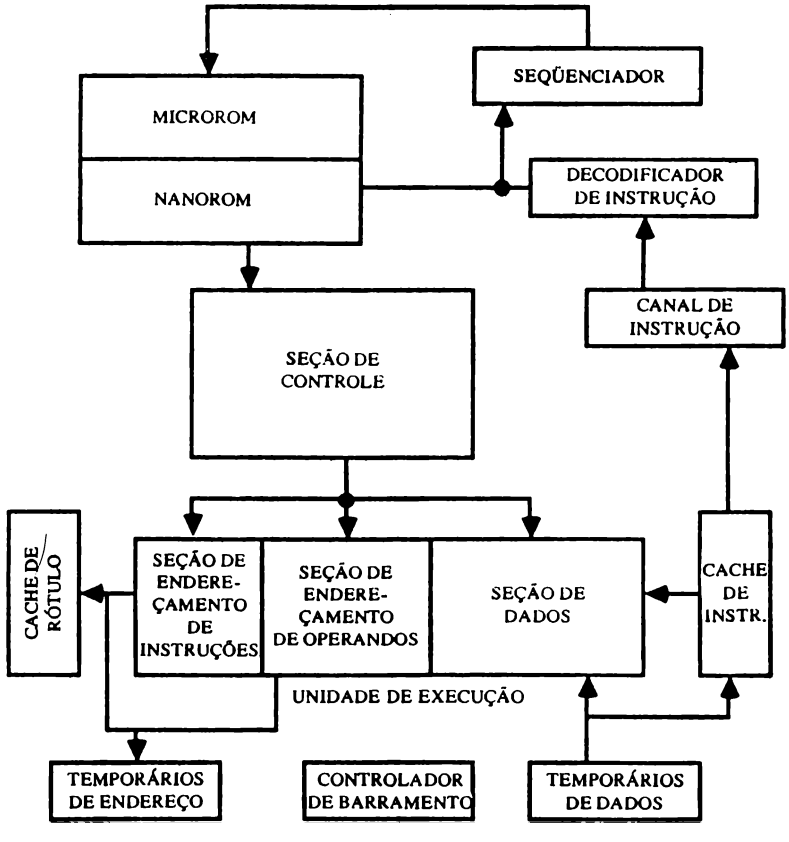


Figura 6.1 Diagrama de blocos funcionais do MC68020.

O controlador de barramento abrange a sinalização de fim de endereço e dados e os multiplexadores necessários para o suporte ao dimensionador de barramento dinâmico. Em adição, o controlador de barramento contém o cache de instruções e o circuito de controle associado, um macrocontrolador de barramento e dois microcontroladores de barramento. O macrocontrolador de barramento escalona o ciclo de barramento em uma base de prioridades, enquanto os dois microcontroladores de barramento controlam os ciclos do barramento; um para acesso a instruções e outro para acesso a operandos.

A micromáquina consiste em um canal de instrução, PAL de decodificaçao, armazenamento de controle ROM, uma unidade de execução e armazenamento de controle misto. A unidade de execução contém uma seção de endereço de instrução e uma seção de dados. O armazenamento de controle ROM inclui a microrom e a nanorom, que controlam o microcódigo. As PAL de decodificação são usadas para prover a informação necessária para o seqüenciamento do microcódigo. O canal de instrução consiste em três estágios (veja Figura 6.2) e provê decodificação de instruções. As instruções que tenham sido lidas do cache em pastilha ou da memória externa são carregadas no estágio B. As instruções são então seqüenciadas do estágio B para o D. O estágio D contém instruções completamente decodificadas, prontas para execução. Se a instrução contém uma palavra de extensão, ela estará disponível no estágio C, no momento em que o código estiver no estágio D.

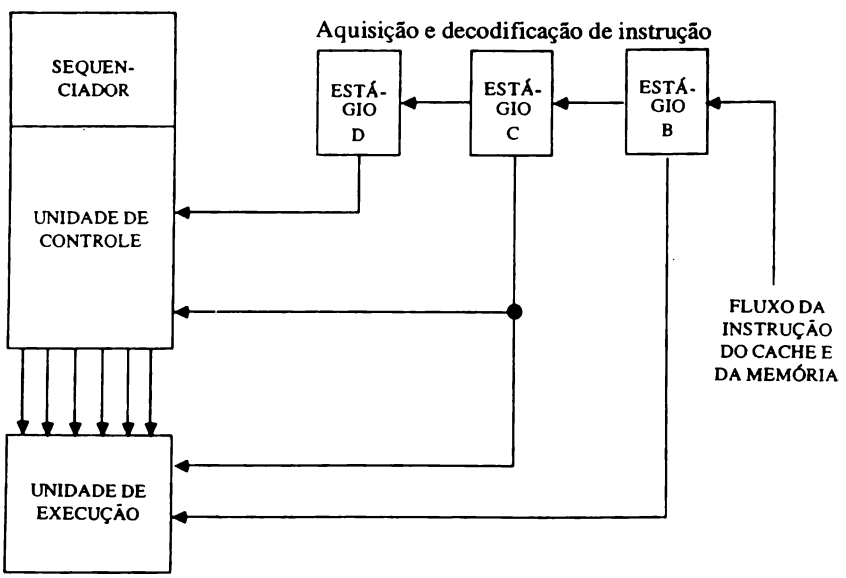


Figura 6.2 Canal de instruções do MC68020.

6.4 ORGANIZAÇÃO DOS DADOS

6.4.1 OPERANDOS

O MC68020 suporta sete tipos de dados básicos: bits, campos de bits, dígitos BCD, inteiros de um byte, inteiros de palavras, inteiros de palavras longas e inteiros de palavras quádruplas. Um campo de bits consiste em 1 a 32 bits, um byte tem 8 bits, uma palavra tem 16 bits, uma palavra longa tem 32 bits e uma palavra quádrupla tem 64 bits. Em adição, o MC68020 suporta operandos de tamanho variável em bytes necessários para a interface do co-processador (isto é, os co-processadores podem definir o tamanho dos operandos requisitados para uma aplicação particular).

6.4.2 MODELO DE PROGRAMAÇÃO

Como nos outros processadores da família M68000, o MC68020 opera em um dos dois modos – usuário ou supervisor. O modo usuário é previsto para prover um meio para programas aplicativos. O modo supervisor tem acesso a instruções adicionais, registradores e privilégios e é previsto para ser usado pelo sistema operacional.

Os modelos de programação usuário e supervisor para o MC68020 são mostrados na Figura 6.3(*a*) e (*b*), respectivamente. O modelo de programação usuário caracteriza-se como 8 registradores de dados de 32 bits, 8 registradores de endereços de 32 bits, um dos quais (A7) é usado como ponteiro da pilha do usuário, um contador de programa de 32 bits e um registrador de código de condições de 8 bits. Em adição aos registradores acima, o modelo de programação supervisor também inclui dois ponteiros de pilha de supervisor de 32 bits (um chamado *mestre* e outro chamado *interrupção*), um registrador de "estado" de 16 bits (o byte menos significativo dele é o registrador de código de condições mencionado acima), um registrador de base de vetor de 32 bits, dois registradores de código de funções alternativos de três bits, um registrador de controle de cache e um registrador de endereço de cache. Uma descrição de cada um desses registradores é dada a seguir.

Os oito registradores de dados (D0-D7) são usados para operações com bits, campos de bits, bytes BCD, palavras, palavras longas e palavras quádruplas. Em geral, as operações em registradores de dados afetam os códigos de condições, mas não fazem com que o resultado no registrador seja estendido por sinal dentro de 32 bits. Os sete registradores de endereço (A0-A6) e os ponteiros de pilha de interrupção, mestre e usuário, podem ser usados como ponteiros de pilha de software, como o registrador de endereço de

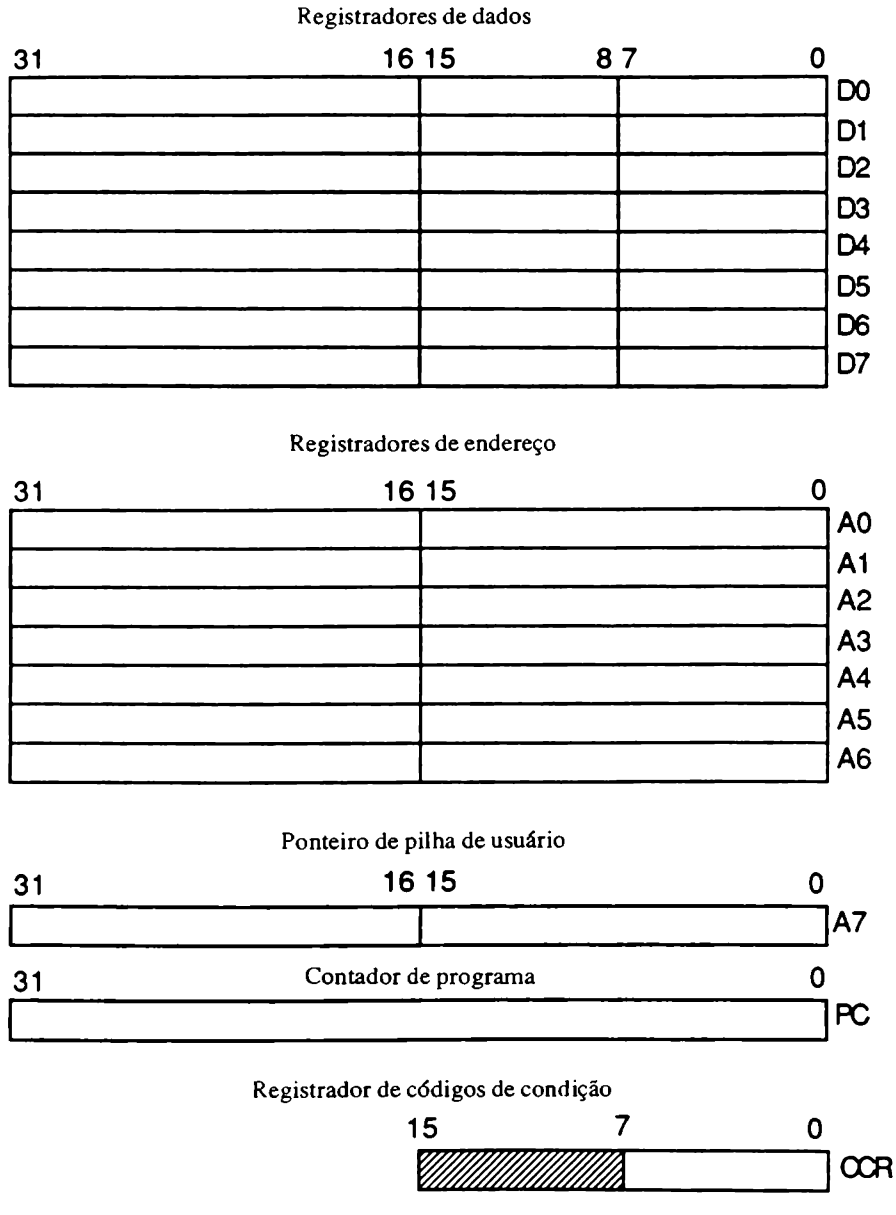


Figura 6.3 (*a*) Modelo de programação de usuário.

base e para operações de palavra e palavra longa. Em geral, as operações em registradores de endereços não afetam os códigos de condições (CMPA é uma exceção) e provocam a extensão do sinal do resultado, nos 32 bits. Além disso, ambos os registradores, de dados e endereços (D0-D7 e A0-A7), podem ser usados como registradores de índice.

O registrador de "estado" (SR) contém 2 bytes: o byte de usuário (registrador de código de condições) e o byte de sistema. O byte de sistema é acessível somente no modo supervisor. O byte de usuário contém bits para as seguintes condições: estendido (X), negativo (N), zero (Z), overflow (V) e carry (C). O byte de sistema tem dois bits de modo "trace" que permitem seguir qualquer instrução ou apenas a mudança de fluxo do programa, o bit S indica se o processador está no modo usuário ou supervisor, o bit M indica se o supervisor está usando a pilha de interrupção ou mestre, e três bits de máscara de prioridade de interrupção. Esses bits fazem o MC68020 ignorar todas as requisições de interrupção com prioridade igual ou menor que a da máscara.

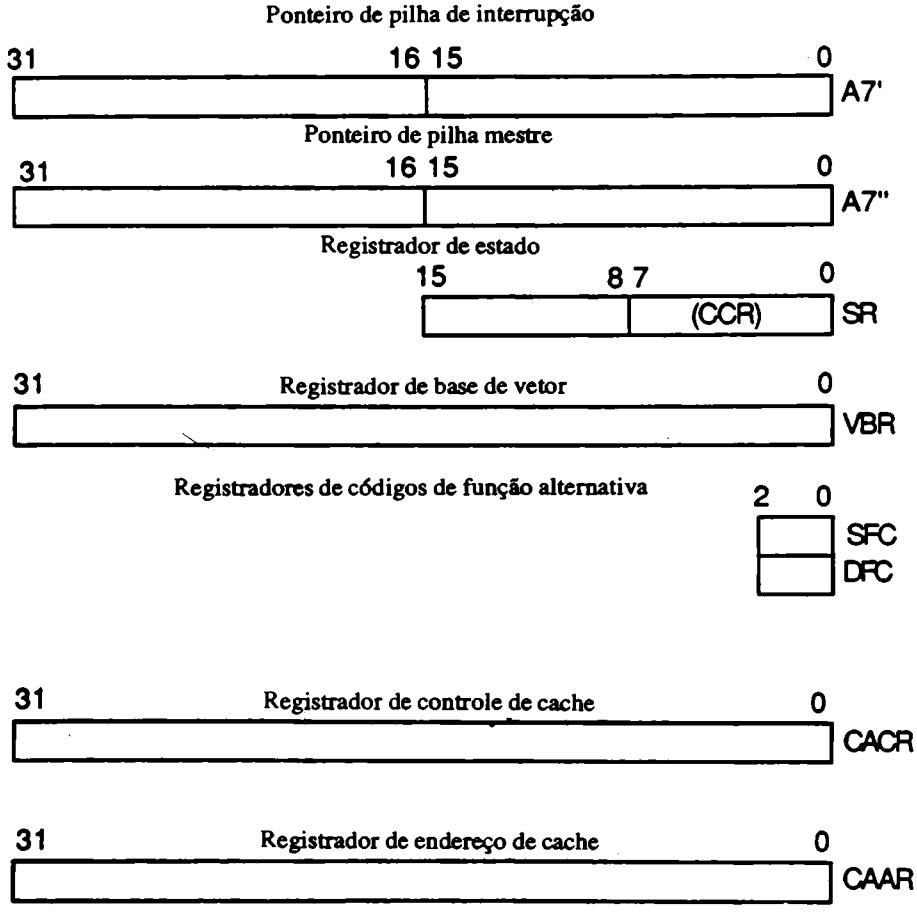


Figura 6.3 (*b*) Suplemento de modelo de programação supervisor.

O MC68020 tem três ponteiros de pilha: usuário, mestre e interrupção. Somente um desses está ativo em um dado momento, dependendo do estado dos bits M e S no re-

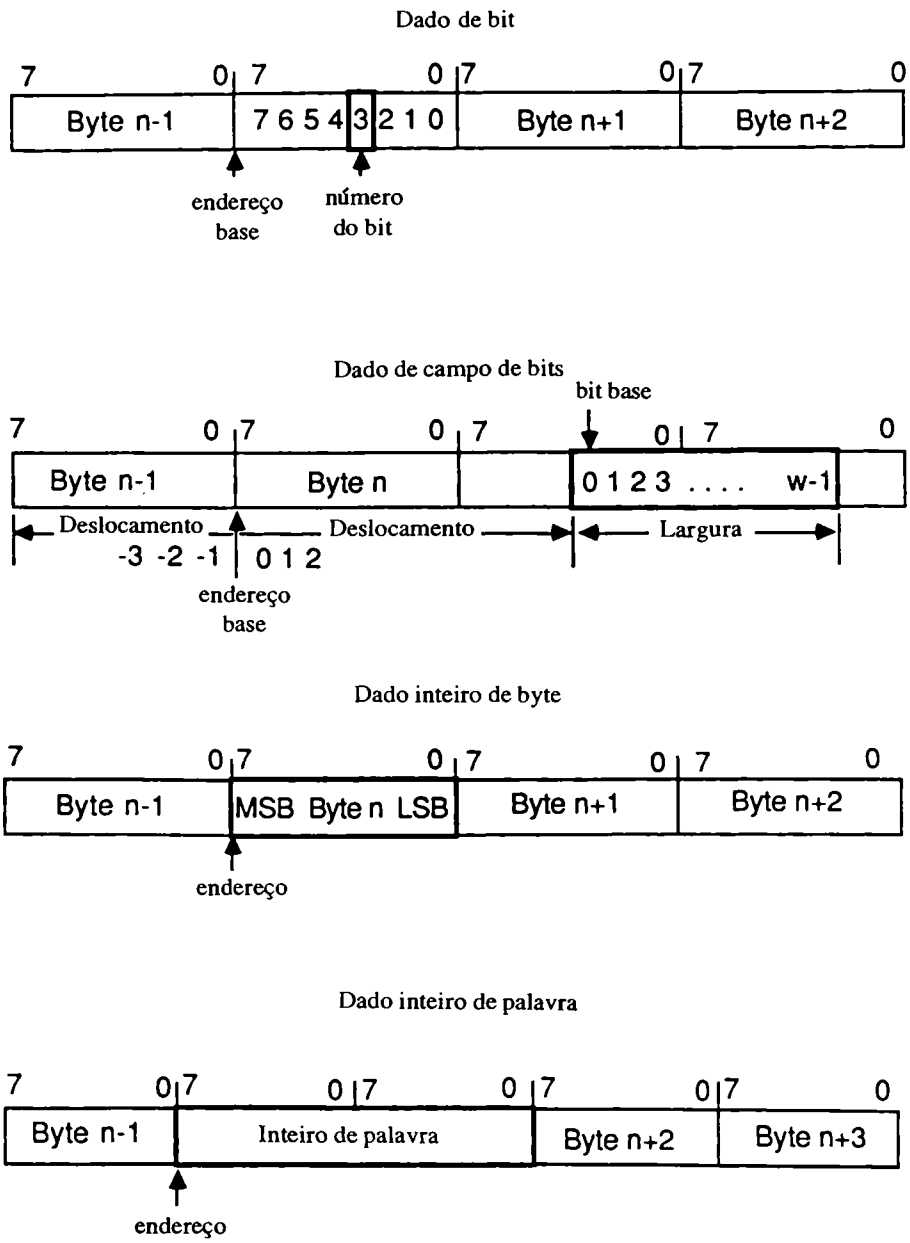


Figura 6.3 (*b*) cont.

gistrador de "estado". O ponteiro de pilha do usuário (USP) opera da mesma forma no MC68020 que nos outros processadores da família M68000. Ele é ativo somente quando o bit S no registrador de estado está limpo. Se ambos os bits M e S estão em 1, o ponteiro de

pilha mestre (MSP) aponta a pilha ativa. Se o bit S está em 1 e o bit M está em 0, o ponteiro de pilha de interrupção aponta a pilha ativa. Este último caso corresponde ao modo supervisor no MC68008, MC68010 e MC68012. As pilhas de interrupção e mestre interagem uma com a outra durante interrupções. Se ambos os bits M e S no registrador de estado são sinalizados (1 – operações na pilha mestre) e uma interrupção é detectada, o deslocamento de vetor de exceção, o contador de programa e o registrador de estado são salvos na pilha mestre. Além disso, o bit M será limpo e uma cópia desse quadro de pilha (chamado *quadro de pilha desperdiçado*), será criado na pilha de interrupção. Se qualquer outra interrupção ocorrer durante o processamento dessa interrupção, a informação de estado correspondente será empilhada somente na pilha de interrupção. Assim, dois ponteiros de pilha trabalham muito bem em ambientes de multitarefa. Cada tarefa pode ter seu próprio ponteiro de pilha mestre e exceções que se referem somente àquela tarefa serão empilhadas ali. Se uma interrupção ocorrer, a informação será empilhada em ambas as pilhas, de interrupção e mestre, porque ela pode referir-se a ambos, tarefa e sistema operacional (como no caso de chaveamento de tarefa). A informação para interrupção que ocorre durante o processamento dessa interrupção seria empilhada somente na pilha de interrupção, porque ela referiria-e somente ao sistema operacional.

O registrador de base de vetor (VBR) contém o endereço base do vetor de exceção de 1 kbyte e é usado para relocar a tabela de vetores para qualquer lugar dentro do intervalo de endereços de 4 gigabytes do MC68020. Como resultado, o MC68020 suporta múltiplas tabelas de vetores tanto quanto poderia ser requisitado em um meio de multitarefa. Quando há reset (reiniciação), o registrador de base ou vetor é zerado.

Os dois registradores de código de funções alternativos (origem, SFC e destino, DFC) permitem que o supervisor acesse qualquer intervalo de endereços (veja Seção 6.4.4) e são usados somente com a instrução MOVES. Em membros mais antigos da família M68000, o MC68000 e o MC68008, os registradores de código de funções alternativos não foram implementados. Assim, todos os acessos do supervisor eram feitos para programa ou dados de supervisor. Quando uma instrução MOVES é executada no MC68020, o conteúdo do registrador de código de função alternativo é colocado nas linhas de código de função (FC0-FC2) para o operando origem ou para o operando destino. Isto permite que o supervisor tenha acesso a outros intervalos de endereço. A instrução MOVES é privilegiada, proibindo, portanto, o usuário de acessar o intervalo do supervisor.

Os dois registradores de cache (controle, CACR, e endereço, CAAR) são providos para permitir manipulação por software do cache de instruções em pastilha. O registrador de controle do cache contém bits para limpar, congelar e habilitar o cache, e limpar uma entrada específica. Quando se limpa uma entrada do cache, o endereço da entrada é apontado pelo registrador de endereço do cache.

6.4.3 ORGANIZAÇÃO DA MEMÓRIA

Como em toda a família do M68000, a memória no MC68020 é organizada de forma que o endereço mais baixo corresponda aos bytes de maior ordem. Por exemplo, o endereço N de uma palavra longa também é o endereço do byte mais significativo da palavra de mais alta ordem. Assim, o endereço do byte menos significativo da palavra de mais baixa ordem é N + 3.

A Figura 6.4 mostra como os tipos de dados suportados pelo MC68020 são organizados na memória. Esses tipos de dados podem ser acessados em qualquer tamanho de byte. O MC68020 não requer que os operandos de dados sejam sequer de tamanho em byte. Essa característica é chamada *não alinhamento* e será discutida na Seção 6.8. Entretanto, o desempenho máximo é alcançado quando os dados são alinhados no mesmo tamanho em bytes que o seu operando. Não é permitido que as instruções sejam não alinhadas. Todas as instruções devem sempre ser alinhadas no mesmo tamanho em bytes. O MC68020 força uma exceção de erro de endereço sempre que o contador de programa contém um endereço ímpar.

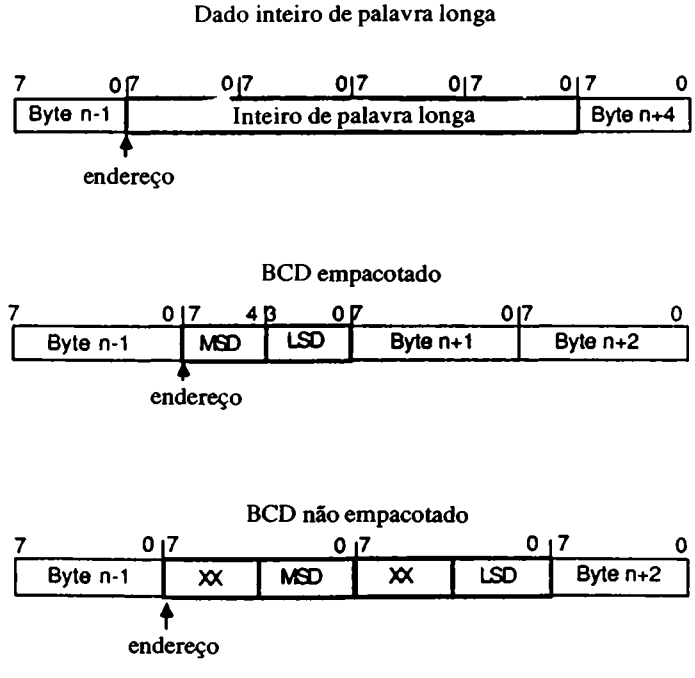


Figura 6.4 Organização de dados na memória.

6.4.4 REFERÊNCIAS A DADOS E PROGRAMAS

A memória é dividida em duas classes no MC68020: programa e dados. A referência a programas refere-se a uma parte da memória que contém instruções. Referência a dados são referências a uma parte da memória que contém dados. Em geral, a leitura de operandos é feita no intervalo de dados e todas as escritas de operandos são feitas no intervalo de dados, exceto quando é feita pela instrução MOVES (move para o intervalo de endereço).

6.5 CAPACIDADE DE ENDEREÇAMENTO

6.5.1 MODOS DE ENDEREÇAMENTO EFETIVO

A localização de um operando usada na execução de uma instrução pode ser especificada em um dos três modos. Primeiro, a localização (ou endereço) pode estar contida em um registrador. Segundo, a instrução, por definição, pode implicar o uso de um registrador específico que contém o endereço do operando. Terceiro, o endereço do operando pode ser especificado por um dos modos de endereçamento restantes. Esses tipos de endereçamento serão discutidos nesta seção.

Modos registrador direto – Esses modos de endereçamento efetivos especificam que o operando está contido em um dos dezesseis registradores de uso geral (A0-A7, D0-D7) ou em um dos seis registradores de controle (SR, VBR, SFC, DFC, CACR e CAAR).

Registrador de dados direto – o operando é encontrado no registrador de dado especificado pelo campo de registrador da palavra de operação

Registrador de endereço direto – o operando é encontrado no registrador de endereço especificado pelo campo de registrador.

Modos registrador indireto – Esses modos de endereçamento efetivos especificam que o operando é localizado em um endereço de memória baseado no conteúdo do registrador de endereço.

Registrador de endereço indireto – o endereço do operando está contido no registrador de endereço especificado pelo campo de registrador.

Registrador de endereço indireto com pós-incremento – o endereço do operando está contido no registrador de endereço especificado pelo campo de registrador.

O conteúdo do registrador de endereço é então incrementado por 1, 2 ou 4, dependendo se o tamanho do operando é de um byte, palavra ou palavra longa. Uma exceção para isso é quando o registrador de endereço é o ponteiro da pilha e o tamanho do operando é um byte. Nesse caso, o ponteiro da pilha será incrementado por dois, para manter o ponteiro da pilha alinhado dentro do mesmo limite de um byte.

Registrador de endereço indireto com pré-decremento – o registrador de endereço especificado pelo campo de registrador é primeiro decrementado por 1, 2 ou 4, dependendo do tamanho do operando. Esse valor decrementado é então usado como o endereço do operando. As pilhas são manipuladas como no parágrafo anterior, com o decremento substituído por incremento.

Registrador de endereço indireto com deslocamento – o endereço do operando é a soma dos conteúdos do registrador de endereço especificado no campo de registrador e um deslocamento estendido por sinal de 16 bits. Esse deslocamento está contido em uma palavra de extensão seguindo a palavra do operando.

Modos de registradores indiretos com índice – Esses modos de endereçamento especificam que o endereço do operando é derivado de um cálculo que envolve o registrador de endereço, um registrador de índice e um deslocamento. O formato do operando de índice é 'Xn.tamanho*Escala'. Xn pode ser qualquer dado ou registrador de endereço. Tamanho especifica o tamanho do índice e pode ser palavra ou palavra longa. A escala permite que o conteúdo do registrador de índice seja multiplicado por 1, 2, 4 ou 8. Operandos de indexação e o deslocamento são sempre estendidos por sinal nos 32 bits antes de serem usados nos cálculos.

Registrador de endereço indireto com índice (deslocamento de 8 bits) – o endereço do operando é a soma dos conteúdos do registrador de endereço especificado pelo campo de registrador, o conteúdo registrador de índice com sinal estendido (tamanho e escala) e um deslocamento com sinal estendido.

Registrador de endereço indireto com índice (deslocamento base) – o endereço do operando é a soma dos conteúdos do registrador de endereço especificado pelo campo de registrador, o conteúdo estendido por sinal do registrador de índice e um deslocamento de 16 ou 32 bits com sinal estendido. Note que esse modo de endereçamento, um modo de endereçamento de "registrador de dados", pode ser obtido usando-se um registrador de dado como um registrador de índice e não especificando um registrador de endereço indireto).

Modos de endereçamento indireto de memória – Esses tipos de endereçamento usam quatro valores especificados pelo usuário na determinação do valor do operando: um re-

gistrador de endereço a ser usado como registrador base; um deslocamento base, o qual é somado ao registrador base; um registrador de indexação; e um deslocamento externo, que é somado ao endereço do operando. Esses modos de endereçamento chamados por endereços intermediários a serem lidos da memória antes de prosseguir com os cálculos de endereços efetivos. O operando de indexação pode ser somado depois que o endereço intermediário for lido (pós-indexação) ou antes que o endereço intermediário seja lido (pré-indexação).

Todos os quatro valores especificados pelo usuário são opcionais. Os deslocamentos base e externo podem ser nulos, palavra ou palavra longa. Se um deslocamento é nulo (ou um elemento é suprimido), seu valor é assumido como zero no cálculo do endereço efetivo.

Pós-indexação indireta da memória – um endereço indireto de memória intermediário é calculado somando-se o registrador base e o deslocamento base. Esse endereço é usado para um acesso indireto à memória de uma palavra longa. Essa palavra longa é então somada ao operando de indexação (Xn.Tamanho*Escala) e ao deslocamento externo, se houver algum, para resultar o endereço efetivo.

Pré-indexação indireta da memória – o operando de indexação (Xn. Tamanho*Escala) é somado ao registrador base e ao deslocamento base. A soma intermediária é então usada para o acesso indireto a uma palavra longa. O deslocamento externo, se houver algum, é então somado à palavra longa para resultar o endereço efetivo.

Modo contador de programa indireto com deslocamento – O endereço do operando é a soma do endereço no contador de programa (PC) e um deslocamento com sinal estendido em 16 bits. O deslocamento é armazenado na palavra extensão do op código. O valor do PC é o endereço da palavra extensão. Todas as referências que usam esse modo de endereçamento são classificadas como referências de programa.

Modos contador de programa indireto com indexação – Esses modos de endereçamento são análogos aos modos registrador indireto com indexação descritos anteriormente, com o PC usado como registrador base. Como antes, o operando de indexação (tamanho e escala) e o deslocamento são usados no cálculo do endereço efetivo. Novamente os deslocamentos e os operandos de indexação são sempre estendidos por sinal em 32 bits para serem usados no cálculo de endereço efetivo.

PC indireto com indexação (deslocamento de 8 bits) – o endereço do operando é a soma do endereço no PC, o inteiro de sinal estendido contido na palavra extensão de 8 bits, e o tamanho e a escala do operando de indexação. O valor no PC é o endereço da palavra de extensão. O usuário deve especificar o deslocamento, o PC e o registrador de indexação quando usar esse modo de endereçamento.

PC indireto com indexação (deslocamento base) – o endereço do operando é a soma do endereço contido no PC, do tamanho e escala do registrador de indexação, e o deslocamento base com sinal estendido. Esse modo de endereçamento requer palavras de extensão adicionais que contenham indicações do registrador de indexação e o deslocamento de 16 ou 32 bits. Todos os três parâmetros são opcionais neste modo.

Modos PC indiretos de memória – Esses modos de endereçamento são análogos aos de endereçamento indiretos de memória descritos previamente, com o contador de programa sendo usado como registrador base. O acesso à memória intermediária pode ser pós-indexado ou pré-indexado. Todos os quatro valores especificados pelo usuário são opcionais.

Contador de programa indireto de memória pós-indexado – um endereço de memória indireto intermediário é calculado com a soma do PC (usado como registrador base) e um deslocamento base. Esse endereço é usado para um acesso indireto de uma palavra longa, seguido pela adição do operando de indexação (tamanho e escala) com a leitura do endereço. Finalmente, o deslocamento externo, se houver algum, é somado para resultar o endereço efetivo.

Contador de programa indireto de memória pré-indexado – o tamanho e a escala do operando de indexação é somado ao contador de programa e ao deslocamento base. Essa soma é usada para acesso indireto a uma palavra longa. O deslocamento externo, se houver algum, é então somado para resultar o endereço efetivo.

Modos de endereçamento absoluto – Esses modos de endereçamento especificam o endereço do operando nas palavras extensão.

Endereço absoluto curto – o endereço do operando está contido na extensão da palavra. O endereço de 16 bits é estendido por sinal nos 32 bits antes de ser usado. Esse modo requer uma palavra de extensão.

Endereço longo absoluto – esse modo de endereçamento requer duas palavras de extensão. O endereço do operando é obtido pela concatenação das palavras de extensão. A primeira palavra de extensão contém a parte mais significativa do endereço, e a segunda palavra de extensão contém a parte menos significativa.

Dado imediato – Esse modo de endereçamento requer uma ou duas palavras de extensão, dependendo do tamanho da operação.

Operação de byte – o operando está no byte menos significativo da palavra de extensão.

Operação de palavra – o operando está nas duas palavras de extensão; os 16 bits mais significativos estão na primeira palavra de extensão; os 16 bits menos significativos estão na segunda palavra de extensão.

6.6 SUMÁRIO DO CONJUNTO DE INSTRUÇÕES

Nesta seção mostraremos o conjunto de instruções do MC68020.

As instruções podem ser classificadas como se segue:

Movimento de dados	Manipulação de campo de bits
Aritmética com inteiros	Aritmética decimal de código binário
Lógica	Controle de programa
Deslocamento de rotação	Controle de sistema
Manipulação de bit	Controle de multiprocessador

Os parágrafos seguintes descrevem o conjunto de instruções do MC68020 por categoria.

6.6.1 MOVIMENTAÇÃO DE DADOS

O meio básico transferência de endereços e transferência e armazenamento de dados é através da instrução de movimentação (MOVE). As instruções de movimento de dados permitem que sejam transferidos operandos em bytes, palavras, palavras longas da memória para memória, da memória para registrador, de registrador para memória, e de registrador para registrador. Instruções de movimento de endereços (MOVE e MOVEA) permitem transferência de operandos de palavra longa e palavra dentro dos limites legais de endereços. Em adição à instrução MOVE, existem várias instruções de movimentação de dados especiais: mover registradores múltiplos (MOVEM), mover dados periféricos (MOVEP), mover rápido (MOVEQ), trocar de registradores (EXG), carregar endereço efetivo (LEA), empilhar o endereço efetivo (PEA), enlaçar a pilha (LINK) e desenlaçar a pilha (UNLK).

6.6.2 OPERAÇÕES DE ARITMÉTICA COM INTEIROS

As instruções de aritmética com inteiros consistem em quatro operações básicas para adição (ADD), subtração (SUB), multiplicação (MULT) e divisão (DIV), assim como

limpar (CLR), negativo (NEG) e comparações aritméticas (CMP, CMPM). Ambas as operações de dados e endereços podem ser executadas com as instruções ADD, SUB e CMP. Operações de endereços são limitadas pelos tamanhos de operandos legais para endereços (16 ou 32 bits), enquanto as de dados podem aceitar todos os tamanhos de operandos.

O MC68020 tem uma instrução de divisão e multiplicação para ambos inteiros sinalizados ou não. A multiplicação de operandos de tamanho palavra resultam em um produto de palavra longa, e a multiplicação de operandos de palavra longa resulta em um produto de palavra longa ou em palavra quádrupla. Um dividendo de palavra longa com um divisor de palavra produz um quociente e um resto de palavra, e uma palavra longa ou palavra quad como dividendo com um divisor de palavra longa produz um quociente e um resto de palavra longa.

O MC68020 tem um conjunto estendido de instruções para implementar operações aritméticas de multiprecisão e tamanhos mistos. Elas são: soma estendida (ADDX), subtração estendida (SUBX), sinal estendido (EXT) e negação de binário com extensão (NEGX).

6.6.3 OPERAÇÕES LÓGICAS

As operações lógicas incluem AND, OR, OR exclusivo (EOR) e complemento de um (NOT). Essas instruções podem ser usadas com todos o tamanhos de operandos de dados. Um conjunto similar de instruções imediatas (ANDI, ORI e EORI) realizam as operações lógicas em todos os tamanhos de operandos imediatos. A instrução de teste (TST) compara aritmeticamente o operando com zero. O resultado é refletido nos códigos de condição.

6.6.4 OPERAÇÕES DE DESLOCAMENTO E ROTAÇÃO

O MC68020 pode executar deslocamento de bits (shifts) em ambas as direções via instruções de deslocamento aritmético (ASL e ASR) e deslocamento lógico (LSL e LSR). Em adição, o processador também tem quatro instruções de rotação (com e sem extensão): ROR, ROL, ROXR e ROXL.

Todas as instruções de rotação e deslocamento podem ser realizadas tanto no registrador de dados como na memória. Deslocamentos de rotação de registros suportam todos os tamanhos de operandos e permitem que um contador de deslocamento seja especificado na palavra de operação da instrução ou em outro registrador. Deslocamentos em

memória e rotação operam em operandos de palavra somente e permitem apenas o deslocamento e a rotação de um único bit.

A última instrução da categoria deslocamento e rotação é a instrução de troca de metades de registrador (SWAP). Essa instrução troca as palavras mais e menos significativas do registrador de dado.

6.6.5 OPERAÇÕES DE MANIPULAÇÃO DE BITS

As instruções de manipulação de bits são teste de bit (BTST), teste de bit e troca (BCHG), teste de bit e limpar (BCLR), e teste de bit e sinalização (BSET). Todas as instruções de manipulação de bit podem ser realizadas tanto na memória como em registradores de dados com o número do bit sendo especificado no campo imediato da instrução ou no registrador de dados. Operandos em registradores são sempre de 32 bits, enquanto operando na memória são de 8 bits.

6.6.6 OPERAÇÕES EM CAMPO DE BITS

As instruções de manipulação de campo de bits são quatro, análogas às de manipulação de bits discutidas na seção anterior, nomeadas, teste de campo de bit (BFTST), teste e troca de campo de bit (BFCHG), teste e limpar campo de bit (BFCLR) e teste de sinalização de campo de bit (BFSET). Em adição, há uma instrução de inserção de campo de bit (BFINS) que insere um valor no campo de bit; extração de campo de bit, sinalizado e não sinalizado (BFEXTS e BFEXTU), que extrai um valor do campo de bit, e encontrar o primeiro campo de bit (BFFFO) que encontra o primeiro bit que seja sinalizado no campo de bit. Essas instruções operam com campos de bits de tamanhos variáveis até 32 bits.

6.6.7 OPERAÇÕES DE DECIMAIS DE CÓDIGO BINÁRIO

Operações aritméticas de multiprecisão e números decimais de código binário (BCD) são feitas através das seguintes instruções: soma de decimal com extensão (ABCD), subtração de decimal com extensão (SBCD) e negação de decimal com extensão (NBCD). As instruções de compactar (PACK) e descompactar (UNPK) são usadas para converter do dado numérico codificado em byte, tal como strings ASCII ou EBCDIC, para BCD e vice-versa.

6.6.8 OPERAÇÕES DE CONTROLE DE PROGRAMA

As operações de controle de programa são executadas usando-se um conjunto de saltos (branch) condicionais e incondicionais e instruções de retorno. As condições em que os saltos podem ocorrer são:

limpar o carry	menor ou igual
sinalizar carry	menor que
igual	menos
nunca verdadeiro	não igual
maior ou igual	mais
maior que	sempre verdadeiro
alto	limpar overflow
menor ou igual	sinalizar overflow

Em adição, essa categoria inclui sempre saltos (branch) (BRA), salto para sub-rotina (BSR), chamada de módulo (CALLM), pulo (jump) (JMP), pulo para sub-rotina (JSR), retorno e desalocação de parâmetros (deallocate) (RTD), retorno de módulo (RTM), retorno e rearmazenamento de condições de códigos (RTR) e retorno de sub-rotina (RTS).

6.6.9 OPERAÇÕES DE CONTROLE DE SISTEMA

As operações de controle de sistema são feitas via instruções privilegiadas, instruções de geração de trap, e instruções que usam ou modificam explicitamente o registrador de código de condições. As instruções privilegiadas realizam o E, OU exclusivo, e OU do dado imediato com o registrador de estado (ANDI, EORI e ORI, respectivamente) como também movimentação de informação do/e para o registrador de estado, do ponteiro de pilha do usuário e dos registradores de controle. Uma outra versão da instrução MOVE, MOVES, usa os registradores do código de função origem e destino para determinar para ou de que intervalos de endereço os dados serão movidos. Outras instruções privilegiadas são RESET (que causa a reiniciação* da linha para ser ativa), RTE (retorno de exceção) e STOP (que faz com que o MC68020 pare de processar a informação). Incluídas na categoria de instruções de geração de trap estão o BKPT (breakpoint – ponto de parada), CHK (check – verificação), CHK2 (verificação 2), ILLEGAL, TRAP, TRAPcc e TRAPV. A instrução BKPT faz com que o MC68020 execute um ciclo de reconhecimento de breakpoint, durante o qual o processador pega os bits de dados D31 até D24 e usa o valor como a palavra op. As instruções CHK e CHK2 comparam o valor de um registrador com algum limite inferior/superior especificado e geram uma exceção ou

continuam a execução da instrução baseados no resultado da comparação. As instruções ILLEGAL e TRAP forçam o MC68020 a gerar uma exceção de instrução ilegal ou uma exceção de trap, respectivamente. O TRAPcc e o TRAPV fazem com que o MC68020 gere um trap se a condição for verdadeira (TRAPcc) ou se um underflow ocorrer (TRAPV). As instruções que afetam o registrador de código de condições são ANDI, EORI, ORI e MOVE. As três primeiras instruções executam E, OU exclusivo, ou OU do dado imediato com o registrador de código de condição, enquanto as últimas instruções movem os dados para e do registrador.

6.6.10 OPERAÇÕES DE CONTROLE DE MULTIPROCESSADORES

É provido suporte a multiprocessadores no MC68020 via instruções TAS, CAS e CAS2. Essas três instruções executam ciclos de barramento de escrita-modificação-leitura indivisíveis. Para indicação do usuário, o sinal RMC* é enviado durante a execução da instrução (veja Seção 6.7).

A instrução de teste e sinalização (TAS) é usada para suportar operações de semáforo. Ela primeiro lê um byte da memória e verifica o bit mais significativo daquele operando, sinalizando os códigos de condição que estejam de acordo. Então ela sinaliza (1) o bit mais significativo daquele byte na memória, independente de seu estado anterior. O programa pode executar um salto (branch) condicional baseado no estado dos códigos de condição. Se o bit foi sinalizado (1) anteriormente, isto indica que outro processador está usando aquele semáforo, e este processador deve esperar antes que ele possa atender aos recursos associados com aquele semáforo. Se o bit foi limpo, o processador executando a instrução TAS já requisitou o uso com a sinalização do bit mais ignificativo. As instruções de comparação e troca (CAS e CAS2) são usadas para atualizar os contadores do sistema, informações históricas e ponteiros de compartilhamento globais. São definidos três operandos para a instrução CAS: o endereço efetivo do operando, o registrador a ser comparado e o registrador de atualização. O operando especificado pelo endereço efetivo é o primeiro a ser lido e comparado com o valor no registrador de comparação. Se os valores forem iguais, o valor contido no registrador de atualização é escrito no endereço efetivo. Se os valores não forem iguais, o valor do operando é carregado no registrador de comparação. A instrução CAS2 trabalha exatamente da mesma forma, exceto que ela usa dois conjuntos de cada operando. Essa instrução é usada para manipulação de listas duplas encadeadas.

Também são incluídas na categoria de operações de multiprocessadores as instruções de co-processadores. Essas instruções serão discutidas em detalhes na Seção 6.10.

6.7 DESCRIÇÃO DOS SINAIS

Esta seção contém uma descrição breve dos sinais de entrada e saída do MC68020. Ele é organizado por funções de sinais, como mostrado na Figura 6.5. O termo "enviado" é usado para indicar que o sinal é ativo, desconsiderando se aquele nível é representado por uma voltagem alta ou baixa. Da mesma maneira, o termo "negativo" é usado para indicar que o sinal está inativo.

Sinais de código de função (FC0-FC2) – Esses sinais de saída de três estados são usados para identificar o estado do processador (supervisor ou usuário) e o intervalo de endereços do ciclo de barramento executando correntemente, como descrito na Tabela 6.1.

Barramento de endereços (A0-A31) – Durante a execução de todos os ciclos de barramento, exceto as referências de intervalo da CPU, essas saídas de três estados provêem o endereço para um ciclo de barramento. Durante as referências de espaço de CPU, o barramento de endereço provê informações relacionadas à CPU. O MC68020 pode endereçar linearmente 4 gigabytes (2^{32}) de dados.

Barramento de dados (D0-D31) – Esses sinais de três estados bidirecionais provêem a passagem de dados entre o MC68020 e outros dispositivos no sistema. O barramento de dados pode transmitir e aceitar dados usando as capacidades de dimensionamento de barramento dinâmico do MC68020.

Tamanho de transferência (SIZ0, SIZ1) – Essas saídas de três estados indicam o número de bytes de um operando que falta ser transferido em um ciclo de barramento dado. Essas saídas são usadas com as capacidades de dimensionamento do barramento dinâmico do MC68020.

Sinais de controle do barramento assíncronos – Os parágrafos seguintes descrevem os sinais de controle do barramento assíncronos no MC68020.

Início do ciclo externo (ECS*). Essa saída provê uma indicação prévia de que o MC68020 está iniciando um ciclo de barramento. Ele é enviado durante a primeira metade do ciclo do relógio de todos os ciclos de barramento. Esse sinal deve ser qualificado futuramente com o AS*, para (insure) certificar a validade do ciclo de barramento do MC68020, desde que o MC68020 possa começar uma leitura de instrução e depois abortar o acesso se a palavra da instrução for encontrada no cache em pastilha. O MC68020 direciona somente o barramento de endereços, sinais de código de função e saídas de dimensão quando ele aborta um ciclo de barramento devido a uma colisão com o cache.

Início do ciclo de operando (OCS*). Esse sinal de saída é enviado somente durante o primeiro ciclo de barramento de uma transferência de operando e tem o mesmo tempo que o sinal ECS*.

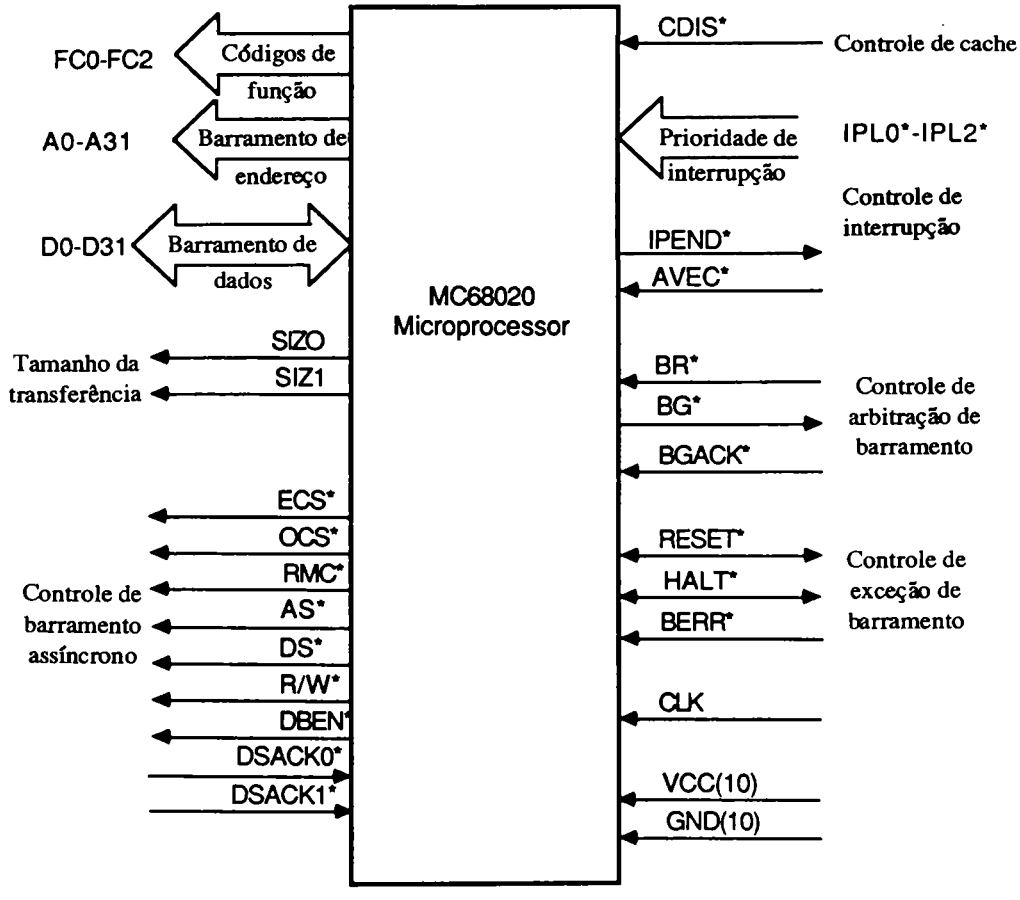


Figura 6.5 Grupos funcionais de sinais.

Tabela 6.1 Atribuição a códigos de função.

FC2	FC1	FC0	Cyclo Type
0	0	0	(Não definido, reservado)*
0	0	1	Espaço de dados de usuário
0	1	0	Espaço de programa de usuário
0	1	1	(Não definido, reservado)
1	0	0	Não definido, reservado
1	0	1	Espaço de dados supervisor
1	1	0	Espaço de programa supervisor
1	1	1	Espaço de CPU

* Espaço de endereço 3 é reservado para definição pelo usuário, enquanto os espaços 0 e 4 são reservados para uso futuro pela Motorola.

Ciclo de leitura-modificação-escrita (RMC*). Esse sinal de três estados é enviado para a duração do ciclo de leitura-modificação-escrita. Ele deve ser usado como um bloqueio de barramento para certificar a integridade das instruções que usam ciclos de leitura-modificação-escrita.

Habilitador de endereço (AS*). Esse sinal de três estados é usado para indicar que endereço válido, código de funções, informação de leitura/escrita está no barramento.

Habilitador de dado (DS*). Esse sinal de três estados é usado para indicar que o periférico selecionado deve direcionar o barramento de dados. Durante o ciclo de escrita, esse sinal indica que o MC68020 alocou um dado válido no barramento de dados.

Leitura/escrita (R/W*). Esse sinal de três estados é usado para indicar a direção da transferência dos dados. Um nível alto indica que um ciclo de leitura está em progresso, e um nível baixo indica que um ciclo de escrita está sendo executado.

Habilitação do buffer de dados (DNEN*). Esse sinal de três estados provê uma habilitação dos buffers de dados externos. Ele permite à linha R/W* transitar sem causar uma contenção de buffer externo. O sinal não é necessário em todos os sistemas.

Reconhecimento de transferência de dados e tamanho (SACKO*, DSACK1*). Essas entradas indicam para o MC68020 que uma transferência de dados está completa e a quantidade de dados que o dispositivo externo o aceitou ou proveu. Durante um ciclo de leitura, quando o MC68020 reconhece o envio do DSACKx*, ele pega o dado e termina o ciclo de barramento. Durante um ciclo de escrita, no reconhecimento do DSACKx*, o processador termina o ciclo de barramento. Mais explicações sobre o DSACKx*, podem ser encontradas na Seção 6.8.

Desabilitador do cache (CDIS*) – Esse sinal de entrada desabilita dinamicamente o cache em pastilha. O cache será desabilitado internamente depois que o sinal CDIS* tenha sido enviado e sincronizado internamente. Veja Seção 6.9.2 para mais informações.

Sinais de controle de interrupções – Os parágrafos seguintes descrevem os sinais de controle de interrupções para o MC68020.

Nível de prioridade de interrupção (IPL0*-IPL2*). Essas entradas indicam o nível de prioridade do dispositivo que requer a interrupção. O nível sete tem a maior prioridade e não é mascarável; o nível zero indica que nenhuma interrupção é requisitada.

Interrupção pendente (IPEND*). Essa saída é usada para indicar que o nível de prioridade nas linhas IPL2*-IPL0* é maior que o nível de prioridade na máscara de interrupção do registrador de estado, ou que uma interrupção não-mascarável foi reconhecida.

Autovetor (AVEC*). Essa entrada pode ser enviada durante um ciclo de reconhecimento, indicando que um número de vetores de interrupção poderia ser gerado internamente.

Sinais de arbitração de barramento – Os parágrafos seguintes descrevem os pinos usados na determinação de quando outros dispositivos podem tornar-se o mestre do barramento.

Requisição de barramento (BR*). Essa entrada é enviada pelo dispositivo indicando que ele deseja tornar-se o mestre do barramento. O sinal poderia ser wire-0Red com os outros sinais de requisição de todos os outros mestres do barramento potenciais.

Garantia de barramento (BG*). Esse sinal de saída indica que o MC68020 liberará o barramento no fim do ciclo de barramento, sendo executado correntemente.

Reconhecimento de garantia de barramento (BGACK*). Esse sinal de entrada indica para o MC68020 que algum outro dispositivo assumiu o comando do barramento. O dispositivo deve manter esse sinal até que ele tenha completado sua transação. Esse sinal poderia ser enviado até que as seguintes condições sejam encontradas:

1. O BG* foi recebido.
2. AS* foi negado, indicando que o MC68020 não está executando um ciclo de barramento.
3. DSACK0* e DSACK1* estão negados, indicando que nenhum outro dispositivo está direcionando o barramento no momento.
4. BGACK* é negado, indicando que nenhum outro mestre do barramento tem o barramento.

Sinais de controle de exceção de barramento – Os parágrafos seguintes descrevem os sinais de controle de exceção de barramento para o MC68020.

Reset (reiniciação) (RESET*). Esse sinal sorvedor aberto e bidirecional é usado como sinal de reset do sistema. Se ele for enviado como entrada, o RESET* fará com que o processador entre em um processamento de exceção de reset. Como uma saída, MC68020 envia RESET* (em resposta à execução da instrução RESET) para reiniciar dispositivos externos, mas não é afetado internamente.

Halt (HALT*). Esse sinal sorvedor aberto bidirecional indica se o processador está no estado de halt ou no estado de execução. Quando HALT* é enviado como uma entrada, MC68020 pára todas as atividades de barramento no fim da execução do ciclo de barramento corrente e coloca todos os sinais de controle no estado em seus estados inati-

Tabela 6.2 Sumário dos sinais.

Função do sinal	*Nome do sinal*	*Entrada/ saída*	*Estado ativo*	*Terceiro estado*
Códigos de função	FC0-FC2	Saída	Alto	Sim
Barramento de endereço	A0-A31	Saída	Alto	Sim
Barramento de dados	D0-D31	E/S	Alto	Sim
Tamanho	SIZ0-SIZ1	Saída	Alto	Sim
Início ciclo externo	ECS*	Saída	Baixo	Não
Início do ciclo de operando	OCS*	Saída	Baixo	Não
Ciclo de leitura – mod. – escrita	RMC*	Saída	Baixo	Sim
Validação de endereço	AS*	Saída	Baixo	Sim
Validação de dados	DS*	Saída	Baixo	Sim
Leitura/escrita	R/W*	Saída	Alto/Baixo	Sim
Habilita buffer de dados	DBEN*	Saída	Baixo	Sim
Transferência/tamanho de dado	DSACK0-1*	Entrada	Baixo	–
Desabilita cache	CDIS*	Entrada	Baixo	–
Nível de prioridade de interrupção	IPL0-IPL2*	Entrada	Baixo	–
Interrupção pendente	IPEND*	Saída	Baixo	Não
Autovetor	AVEC*	Entrada	Baixo	–
Requisição de barramento	BR*	Entrada	Baixo	–
Bus Grant	BG*	Saída	Baixo	Não
Reconhecimento de Bus Grant	BGACK*	Entrada	Baixo	–
Reset	RESET*	E/S	Baixo	Não*
Suspenso	HALT*	E/S	Baixo	Não*
Erro de barramento	BERR*	Entrada	Baixo	–
Relógio	CLK	Entrada	–	–
Alimentação	VOC	Entrada	–	–
Terra	GND	Entrada	–	–

* Dreno aberto

vos. O processador irá, entretanto, continuar a direcionar as linhas de código de função e o barramento de endereços.

MC68020 envia HALT* como um resultado de uma condição de erro barramento duplo para indicar aos outros dispositivos no sistema que ele parou de executar instruções.

Erro de barramento (BERR*). Este sinal de entrada é usado para indicar ao MC68020 que há um problema com o ciclo de barramento sendo executado correntemente. Esses problemas poderiam ser o resultado de:

1. Dispositivos não respondendo.
2. Falha na aquisição de um vetor de interrupção.

3. Acesso ilegal como determinado pela unidade de gerenciamento de memória.
4. Erros dependentes de outras aplicações.

O sinal BERR* interage com o sinal HALT* para determinar se o ciclo de barramento corrente poderia ser reexecutado ou abortado.

CLOCK (CLK) – Esse sinal compatível com TTL é vetorizado internamente para gerar os relógios internos requisitados pelo MC68020. Ele poderia não ser uma porta a qualquer momento.

Sumário de sinais – A Tabela 6.2 provê um sumário dos sinais do MC68020 discutidos nos parágrafos anteriores.

6.8 MECANISMO DE TRANSFERÊNCIA DE OPERANDOS

6.8.1 DIMENSIONAMENTO DO BARRAMENTO DINÂMICO

O MC68020 suporta um mecanismo de transferência de operando chamado *dimensionamento do barramento dinâmico* que lhe permite determinar o tamanho da porta (8, 16 ou 32 bits) em uma base ciclo de barramento por ciclo de barramento. Durante o ciclo de barramento, uma porta irá sinalizar seu tamanho de barramento de dados e transferir o estado (completo ou não) para o MC68020 via linhas de entrada DSACK0* e DSACK1*. Essas entradas DSACKx* têm a mesma função que a entrada DTACK* dos outros processadores da família M68000, assim como indica o tamanho da porta. A Tabela 6.3 descreve os códigos e meios do DSACKx*.

Como um exemplo, se o MC68020 tem que ler um valor de 32 bits de uma porta, ele não precisa saber o tamanho da porta antes de começar a transferência. Se a porta responde com um DSACK1*/DSACK0* = 00, indicando que ela é uma porta de 32 bits, o MC68020 pega todos os 32 bits do barramento de dados e completa a transferência (isto é, nega os sinais de habilitação de endereço e de dados e controle). Se a porta indica que ele é de 16 bits (DSACK1*/DSACK0* = 01), o processador pega 16 bits no D31-D16 e começa outro ciclo de barramento. Quando a porta envia o DSACKx*, o MC68020 pega novamente o dado entre D31-D16, e assim lê todos os 32 bits. Eventos similares poderiam ocorrer se a porta fosse de 8 bits, exceto que seriam necessários quatro ciclos de barramento para ler em quatro bytes nos D31-D24.

A fim de alocar dados válidos, o MC68020 faz certas suposições sobre onde as portas são alocadas com relação a seu barramento de dados. As portas de 32 bits deveriam

residir em D31 a D0; as de 16 bits, em D31 a D16; as de 8 bits, em D31 a D24. Tendo-se as portas alinhadas dessa maneira, o número de ciclos requeridos para a transferência de dados é minimizado.

Tabela 6.3 Códigos $\overline{\text{DSACK}}$ e resultados

$\overline{\text{DSACK1}}$	$\overline{\text{DSACK0}}$	Resultado
H	H	Insira ciclos de espera do ciclo de barramento corrente
H	L	Complete ciclo – Porta do barramento de dados é 8 bits
L	H	Complete ciclo – Porta do barramento de dados é 16 bits
L	L	Complete ciclo – Porta do barramento de dados é 32 bits

O MC68020 contém um multiplexador interno que lhe permite rotear os quatro bytes do barramento de dados para suas posições corretas. Esse multiplexador provê o mecanismo pelo qual o MC68020 suporta o dimensionador de barramento dinâmico e operandos não alinhados. Por exemplo, o byte mais significativo do operando, OP0, pode ser roteado para os bits D31 a D24, como em operações normais (alinhadas), ou ele pode ser roteado para qualquer outra posição para suportar o não alinhamento. O mesmo se aplica pra outros bytes de operandos. A posição que o operando ocupa é determinada pelas linhas de tamanho (SIZ0 e SIZ1) e as duas linhas de endereço menos significativas (A1 e A0).

As linhas SIZ1 e SIZ0 indicam o número de bytes restantes que são realmente transferidos; dependendo do tamanho da porta, o alinhamento do operando será menor ou igual ao valor indicado pelas linhas SIZx. A Tabela 6.4 mostra os códigos de saída do SIZx.

As linhas de endereço A1 e A0 também afetam a operação do multiplexador. Durante a transferência do operando, A31 e A2 dão o endereço base da palavra longa do operando a ser acessado, e A1 e A0 dão o byte de deslocamento daquele endereço. A Tabela 6.5 mostra os códigos de deslocamento de endereço.

A Tabela 6.6 descreve o uso das linhas SIZ0, SIZ1, A0 e A1 na definição de transferência do multiplexador interno do MC68020 para o barramento de dados externo. Por exemplo, o MC68020 precisa mover o valor $01234567 para uma porta de 8 bits alocada no endereço $8000. Quando o processador começa a executar a instrução, ele não sabe para que tamanho de porta está sendo enviada a informação, então se dedica a escrever todo o operando, nesse caso 32 bits, durante o primeiro ciclo de barramento. As linhas de tamanho são ambas 0, para indicar que a palavra longa é alinhada. A porta de 8 bits pegará bits em D31 a D24 e enviará um DSACK1*/DSACK0* de 10, primeiro para indicar

que ele recebeu os dados, e, segundo, para avisar ao MC68020 que é uma porta de somente 8 bits. Como um resultado de um DSACK' de 8 bits, o MC68020 inicia outro ciclo de barramento com o próximo byte mais significativo de dados multiplexado com o byte último de seu barramento de dados (D31-D24). As linhas de tamanho indicam que os três bytes que restam para serem transferidos e A1 e A0 são incrementados por um para indicar que o próximo byte no mapa da memória está sendo endereçado. Novamente a porta pega em D31 a D24 e sinaliza de volta um DSACK de 8 bits. Esse processo continua até que uma palavra longa seja transferida por completo.

Tabela 6.4 Código de saída SIZE

SIZ1	SIZ0	Size
0	1	Byte
1	0	Palavra
1	1	3 Bytes
0	0	Palavra Longa

Tabela 6.5 Codificação de deslocamento de endereço

A1	A0	Offset
0	0	+ 0 Bytes
0	1	+ 1 Byte
1	0	+ 2 Bytes
1	1	+ 3 Bytes

6.8.2 NÃO ALINHAMENTO DE OPERANDOS

O MC68020 também suporta transferência de operandos de dados não alinhados. A transferência de um operando de/para a memória é considerada como não alinhada se o endereço não cair dentro dos limites de tamanho do operando. A transferência de uma palavra para um endereço ímpar ou uma palavra longa para alguma coisa que não seja da dimensão de palavra longa são exemplos de transferência não alinhada.

Tabela 6.6 Multiplexador do barramento de dados interno com externo do MC68020

Tamanho da transferência	Tamanho		Endereço		Fonte/destino Conexão ao barramento de dados externo			
	SIZ1	SIZ0	A1	A0	D31:D24	D23:D16	D15:D8	D7:D0
Byte	0	1	x	x	OP3	OP3	OP3	OP3
Palavra	1	0	x	0	OP2	OP3	OP2	OP3
	1	0	x	1	OP2	OP2	OP3	OP2
3 Byte	1	1	0	0	OP1	OP2	OP3	OP0
	1	1	0	1	OP1	OP1	OP2	OP3
	1	1	1	0	OP1	OP2	OP1	OP2
	1	1	1	1	OP1	OP1	OP2	OP1
Palavra longa	0	0	0	0	OP0	OP1	OP2	OP3
	0	0	0	1	OP0	OP0	OP1	OP2
	0	0	1	0	OP0	OP1	OP0	OP1
	0	0	1	1	OP0	OP0	OP1*	OP0

* Em ciclos de escrita, este byte é saída; em ciclos de leitura, este byte é ignorado.
x = sem efeito

O MC68020 não coloca restrições em alinhamento de dados. Entretanto, alguma degradação de desempenho pode ocorrer devido aos ciclos de barramento extras que o MC68020 deve executar quando os acessos a palavra ou palavra longa não são feitos den-

tro dos limites de palavra e palavra longa. Note que as instruções devem sempre residir dentro de limites de byte para certificar a compatibilidade com outros membros da família M68000 e para otimizar o desempenho de leitura de instruções. Em qualquer momento que uma leitura de instrução é feita em um endereço ímpar, o MC68020 força uma exceção de erro de endereço. Isto ocorre em qualquer momento em que uma instrução deixar o contador de programa sinalizado com um endereço ímpar.

Como um exemplo de leitura não alinhada, suponha que o MC68020 está lendo uma palavra do endereço $4001 e armazenando-a no registrador de dado D0, onde o endereço $4001 corresponde a uma porta de 16 bits. As linhas SIZ1 e SIZ0 dão 1 e 0, respectivamente, indicando que a palavra precisa ser transferida. A porta de 16 bits escreve a palavra no endereço $4001 ($0123) nos bits 31 a 16 do barramento de dados do MC68020 e sinaliza de volta um DSACK1*/DSACK0* de 01. Esse código indica ao processador que um dado válido está presente no barramento de dados e que o tamanho da porta é de 16 bits. O MC68020 pega o dado, ignora todos os bytes, menos os que estão em D23 a D16, e os armazena ($23) em um registrador temporário. Depois ele executa outro ciclo de barramento, com as linhas de tamanho indicando uma transferência de dados, e o endereço incrementado por um. A porta de 16 bits então escreve a palavra no endereço $4002 (4567) e retorna um DSACK* de 16 bits. O MC68020 pega o dado, ignora os bits de 23 a 0 e armazena o valor de D31 a D24 ($45), junto com o byte armazenado no registrador temporário no ciclo anterior, na palavra menos significativa do registrador de dados D0.

6.8.3 VANTAGENS DO DIMENSIONAMENTO DE BARRAMENTO DINÂMICO

A capacidade do dimensionamento do barramento dinâmico do MC68020 dá aos projetistas de sistemas uma flexibilidade considerável; eles podem escolher o tamanho das portas no sistema como desejarem. Por exemplo, em sistemas em que a RAM seja acessada numerosas vezes, então transferências (dela e para ela) poderiam ser minimizadas para obter um desempenho máximo. Assim, em um sistema MC68020, a RAM é tipicamente de 32 bits. Entretanto, o mesmo não é necessariamente verdade para a ROM. Desde que as EPROM existentes são somente de 8 bits, as portas de ROM de 16 bits requerem partição do programa ROM em bytes ímpares e pares, e portas de ROM de 32 bits requerem partição do programa em quatro pedaços. Se a ROM contém uma rotina monitora onde a velocidade de execução não é o fator maior no desempenho do sistema, então o projetista poderia escolher fazê-lo de somente 8 bits. Isto poderia eliminar a necessidade de particionar a rotina em dois ou quatro bytes toda vez que uma nova EPROM precisasse ser queimada. Também, as mudanças na rotina seriam mais fáceis e rápidas. Os programadores são livres de qualquer constrangimento impostos pelo hardware. Porque cada porta tem um único

acesso via as linhas DSACK*, o sistema, em essência, torna-se "independente do software". Em outras palavras, o programador pode enviar qualquer tamanho de dados para qualquer porta do sistema.

6.9 CACHE

6.9.1 FUNDAMENTOS DO CACHE

A memória cache diferencia-se da memória principal em alguns aspectos. Primeiro, o cache é menor e tem um tempo de acesso muito mais rápido que a memória principal. Segundo, o processador acessa a memória cache de uma forma diferente da memória principal. Quando a memória principal é acessada pelo processador, ele escreve os valores dos dados em endereços específicos. A memória cache, entretanto, deve primeiro comparar o endereço de entrada com o endereço (ou endereços) armazenados no cache. Se os endereços combinam, então diz-se ter ocorrido um *hit* (colisão) e o correspondente está liberado para ser lido do cache. Se o endereço não combina, diz-se ter ocorrido uma "falta" e é permitido um acesso da memória principal para completar. Quando uma falta ocorre, o dado recuperado pela memória principal é também provido para o cache, para que da próxima vez que esse endereço específico for acessado, uma colisão possa ocorrer no cache.

As seções seguintes descrevem os tipos de cache que são ou poderiam ser implementados com o MC68020. A primeira seção discute o cache em pastilha do MC68020 e a segunda cobre os dois modos de projetar um cache de dados externo para o sistema MC68020.

6.9.2 EM PASTILHA

O MC68020 contém um cache de instrução em pastilha que provê desempenho do microprocessador diminuindo o tempo de acesso a instruções e reduzindo as atividades externas do barramento externo do processador.

As palavras de instruções que estão armazenadas no cache são acessadas muito mais rapidamente do que se tivessem sido armazenadas na memória externa. Em adição, enquanto o MC68020 está acessando uma instrução do cache, ele pode fazer uma leitura de dados simultânea no barramento externo.

Um melhoramento na banda do barramento pode ser obtido usando-se o cache

Se o MC68020 encontra uma palavra de instrução no cache, ele não precisa fazer acessos externos, liberando, portanto, o barramento para outros mestres do barramento no sistema.

O cache em pastilha do MC68020 é um cache mapeado diretamente e contém 64 entradas de palavra longa. Cada entrada consiste em um campo de identificação, um bit de validade e palavras de instrução (32 bits). O campo de targeta comprime os 24 endereços mais altos e o valor de FC2. Assim o intervalo de endereços de 4 gigabytes do MC68020 é particionado em blocos de 256 bytes.

Sempre que o MC68020 faz uma leitura de instrução, ele começa dois acessos simultâneos, o normal na memória externa (assumindo-se que o barramento externo está livre) e no cache (assumindo-se que o cache está habilitado). O processador verifica o cache para ver se a instrução está presente. Para fazer isto, uma porção do campo da targeta é usada como indexador do cache para selecionar uma das 64 entradas. Depois, o restante da targeta para aquela entrada é comparado ao endereço da instrução e à linha FC2. Se elas concordam e o bit de validade está sinalizado, diz-se que ocorreu uma colisão. O bit A1 é usado para selecionar a palavra apropriada do cache e o fim do ciclo. Isto tudo ocorre dentro de dois ciclos, em oposição ao normal de três ciclos requisitados para um acesso externo. Externamente, o processador pode direcionar as linhas de endereços, tamanho, código de função e as linhas ECS*, antes de abortar o ciclo devido a uma colisão de cache. Note que o habilitador do endereço não é enviado se houver uma colisão no cache.

Se a identificação da entrada não combina com o endereço da instrução ou se o bit de validade é zero, ocorre uma falta e é permitido que o ciclo externo seja completado. Se o cache não estiver congelado (via o bit congelador no registrador de controle do cache), a nova instrução é escrita no cache e o bit de validade é setado. Note que ambas as palavras correspondentes àquele endereço são realocadas, porque o MC68020 sempre lê palavras longas.

Durante uma reiniciação (reset), o processador limpa o cache limpando os bits de validade para cada entrada. Em adição os bits de habilitação e congelamento no registrador de controle do cache são limpos.

6.9.3 EXTERNO

Um cache externo pode ser projetado para um sistema MC68020 para prover ainda mais desempenho. O cache pode ser do lado lógico ou físico do barramento de endereços e pode ser implementado como cache de instruções e dados ou somente dados.

Discutiremos a seguir os pontos importantes em projetar-se um cache de dados lógico para um sistema MC68020. Isso não significa que será provido ao leitor um projeto testado ou que implicará a melhor ou única forma possível de projeto. Ao contrário, ele poderia ser usado para determinar que especificações de tempo são importantes quando se projeta um cache externo para um sistema MC68020.

Tipicamente, para a memória principal, as especificações de tempo críticas são para endereços válidos para dados válidos. Para uma memória cache, a especificação de tempo mais importante para se minimizar estados de espera é a de endereços válidos para requisições de decisões (colisão ou falta). A decisão poderia causar um evento que causasse o término do ciclo de barramento. Nos sistemas MC68020, o evento poderia ser tanto um envio de DSACK* como o envio de um BERR* ou HALT. Cada um desses métodos será visto, incluindo informação de tempo e vantagens e desvantagens.

No primeiro método, o envio de DSACKx* é retardado até que seja conhecido se uma colisão ocorreu ou não no cache. Assim, um DSACKx* é um produto de um sinal HIT* do circuito do cache. Se ocorre uma colisão, o acesso à memória principal (que pode ocorrer simultaneamente com o acesso ao cache) deve ser abortado antes que a memória possa iniciar o comando do barramento de dados. Se ocorrer uma falta, então será permitido ser completado um ciclo para a memória principal. A Figura 6.6 mostra o pior caso de requisição de tempo a 16,67 MHz para um projeto de cache sem estados de espera (assu-

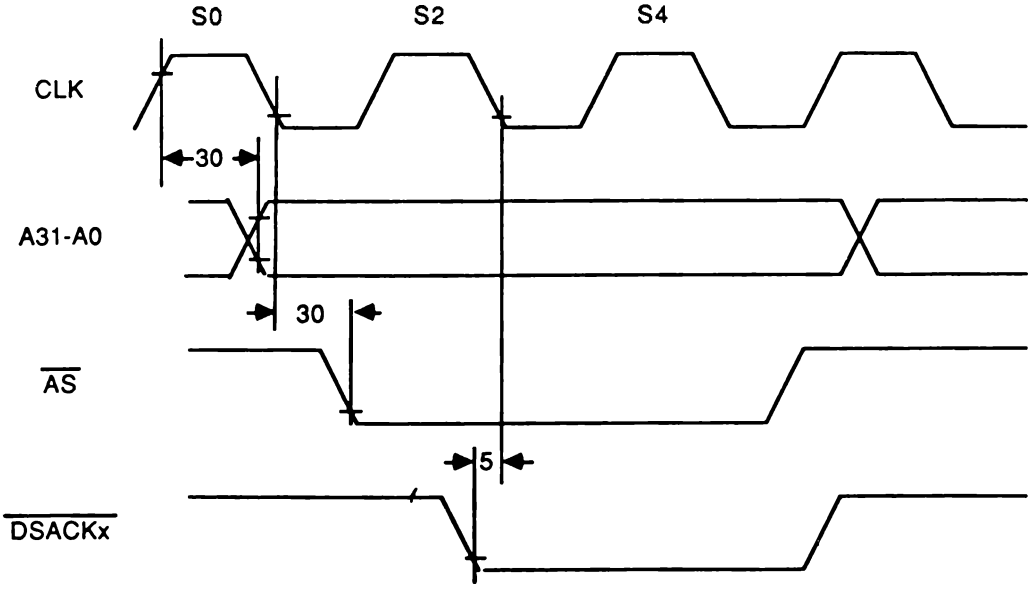


Figura 6.6 Temporização do cache (Caso 1).

mindo uma colisão) usando esse método. Note que a quantidade de tempo disponível para determinar se haverá uma colisão ou uma falta é de 5 nanossegundos mínimo (o tempo para um endereço válido para um envio de DSACK*). Se uma colisão ocorre, os dados precisam estar disponíveis no mínimo 55 nanossegundos antes da borda de descida de S4. Se o projetista não pretende investir em RAM rápidas necessárias para um cache sem estados de espera, estes poderão ser somados para um tempo de decisão mais longo com degradação do desempenho.

No segundo método, é assumido que o cache terá sempre de corrigir os dados (isto é, nunca ocorrerá uma falta). Portanto os DSACKx* podem ser enviados antes que seja conhecido se o endereço está disponível no cache. Se ocorrer uma colisão, as linhas BERR* e HALT* para o MC68020 deverão ser sinalizadas para indicar ao processador que o ciclo não deve terminar normalmente. Isto é conhecido como *late retry*. Quando BERR* e HALT* estão negados, o MC68020 roda novamente o ciclo de barramento que falta. É necessário algum hardware externo para impedir a ocorrência de acesso ao cache nesse segundo ciclo de barramento e para manipular o caso onde o árbitro do barramento ocorra entre os dois ciclos de barramento. Um projeto inteligente tem a vantagem de tempo entre esses ciclos de barramento (falta e retentativa) e inicia o acesso à memória principal quando torna-se óbvio que faltará um acesso ao cache. Veja a Figura 6.7 para requisi-

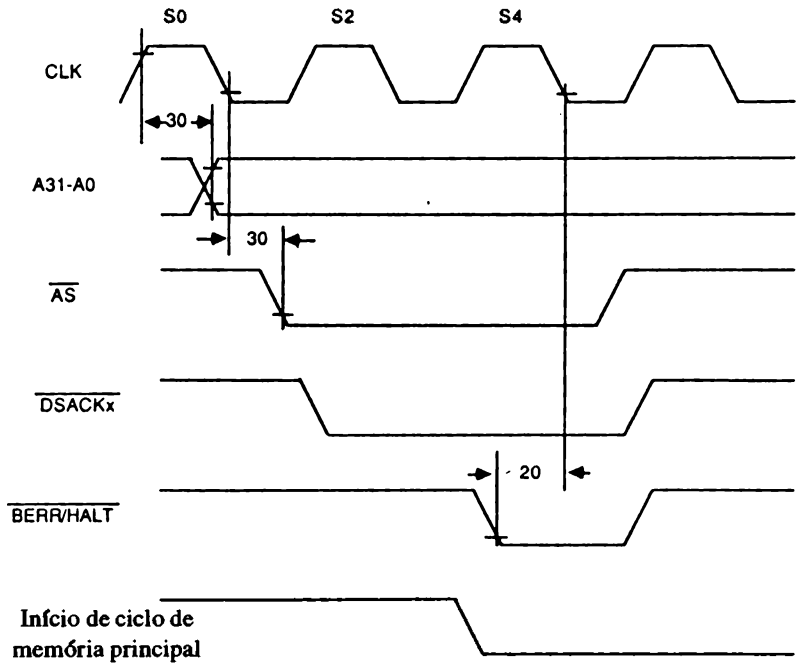


Figura 6.7 Temporização do cache (Caso 2).

ções de tempo (assumido e falta) para esse tipo de projeto de cache. A única requisição para sinalização do DSACKx* é que ele esteja presente no mínimo 5 nanossegundos antes da borda de descida de S2. BERR* e HALT* precisam ser sinalizados no mínimo 20 nanossegundos (de endereços válidos para a sinalização de BERR*/HALT*) para determinar se uma colisão no cache irá ocorrer. Se for determinado que uma falta irá ocorrer, os drivers de barramento de dados das RAM de dados precisarão ser bloqueados para impedi-las de direcionar o barramento, e o acesso à memória principal é iniciado aproximadamente ao mesmo tempo que as linhas BERR* e HALT*.

No primeiro tipo de projeto, o cache requer RAM mais rápidas por não executar estados de espera. No segundo tipo, podem ser usadas RAM mais vagarosas, obtendo-se o mesmo desempenho. O acesso ao cache nesse segundo caso sempre executa com zero estado de espera, e o acesso à memória devido à falta de cache efetivamente roda com dois estados de espera a menos que o acesso à memória principal normal o faça (isto é, sistema sem cache).

6.10 CO-PROCESSADORES

6.10.1 CO-PROCESSADOR INTERFACE

O MC68020 suporta interface de co-processadores que lhe permite estender suas funções. Enquanto uma máquina de proposta geral executa bem em várias áreas diferentes da aplicação, ela poderia não executar tão bem em uma área como um processador projetado especificamente para aquela aplicação. Os co-processadores permitem que o processador principal seja direcionado para uma aplicação particular sem perder a generalidade da arquitetura de processador principal.

Em adição, os co-processadores permitem que os projetistas de sistemas com seus projetos por encomenda, selecionem somente aqueles processadores que preencham as necessidades de seus sistemas. Isto elimina a necessidade dos projetistas de pagarem por um hardware extra que eles freqüentemente têm de comprar para obter o hardware específico que precisam.

Um co-processador é definido como qualquer coisa que implemente a interface de co-processador, ou seja, ele é um dispositivo, vários dispositivos, ou toda uma placa. Ele provê extensões para o modelo de programação do processador principal somando novos registradores e novas instruções e tipos de dados.

A implementação de co-processadores é completamente transparente para o

usuário. O programador não precisa ter conhecimento do protocolo de co-processador. De fato, ele não precisa nunca saber que o co-processador é separado do MC68020. Ele aparece como uma extensão do hardware do MC68020.

A comunicação entre o MC68020 e o co-processador é iniciada pelo MC68020 como resultado de uma instrução de co-processador. O processador começa escrevendo um comando para o co-processador e esperando por uma resposta. O algoritmo necessário para essa comunicação está contido no microcódigo do MC68020. Assim, o usuário não é responsável pelo envio de palavras de comando e *polling* por uma resposta do co-processador. Somente é necessário usar as instruções de co-processador, designadas pelos "uns" no bit mais significativo do código (bits 15-12 = 1111). A Figura 6.8 mostra o formato de uma palavra de instrução de co-processador ou palavra F-linha. Os bits 11 a 9 são definidos como CP-ID (campo de identificação de co-processador – coprocessor identification field). Cada processador no sistema tem um único ID. Assim, mais que oito co-processadores são suportados no sistema. Quando o processador principal inicia a comunicação com o co-processador, o CP-ID é alocado nas linhas de endereço A15-A13. Enquanto o co-processador não necessariamente pode decodificar esses bits internamente, eles podem ser usados dentro do decodificador de endereços para prover selecionamento de chips para o co-processador. Os bits 8 - 6 especificam o tipo de instrução que está sendo executada: geral, salto (branch), condicional, salvamento. rearmazenamento. Esses tipos de instruções serão discutidos futuramente.

Um co-processador também pode ser usado como um periférico com um microprocessador que não o MC68020, tal com o MC68000, o MC8008 ou o MC68010. Se uma instrução F-linha é executada em um sistema M68000, não o MC68020, o processador tem a execução F-linha, assim permite que a interface de co-processador seja emulada em software.

A interface de co-processador opera com ciclos de barramento normal do M68000; não são necessários sinais especiais para conectar o co-processador ao processador M68000. Quando executando como co-processador com o MC68020, as linhas de código de função, ao longo das linhas de endereço A11 - A9, são usadas para gerar a seleção de chip para o co-processador. Todos os acessos do co-processador são feitos no espaço da CPU. Quando acessado como periférico, é necessário gerar-se uma seleção de ship baseada na linha de endereços, tal como qualquer outro periférico. As Figuras 6.9 (*a*) e (*b*)

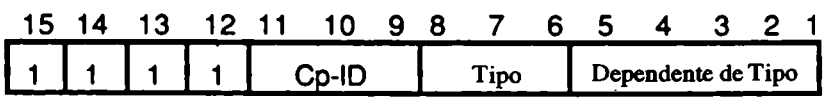


Figura 6.8 Palavra de instrução de co-processador operação F-linha

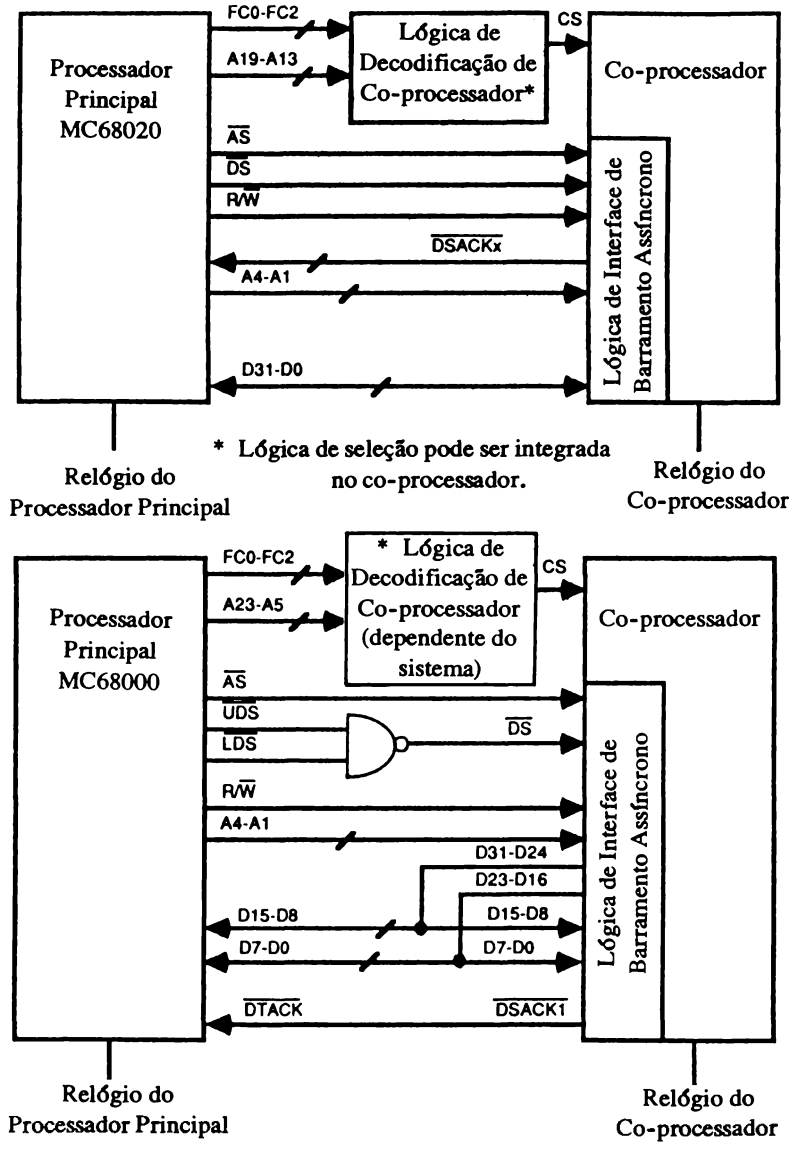


Figura 6.9 (*a*) Interface do MC68020 com co-processadores em barramento de 32 bits; (*b*) Interface do MC68020 com co-processadores em barramento de 16 bits.

mostram um diagrama de blocos de um co-processador em um sistema MC68020 e um M68000, respectivamente. Note que o processador principal e o co-processador não precisam rodar o mesmo relógio nem precisam na mesma velocidade do relógio.

Um processador comunica com o co-processador via registradores de interface

do co-processador, mostrados na Figura 6.10. Aqueles com asteriscos indicam os registradores que são necessários para implementar cada um dos tipos de instrução. Note que todos os endereços estão no espaço da CPU. A seguir descreveremos o conjunto de registradores de interface de co-processador.

Registrador de resposta – um registrador de 16 bits somente para leitura pelo qual o co-processador requisita ação do processador principal.

Registrador de controle – registrador de 16 bits somente para escrita pelo qual o processador central reconhece o processo de exceção de requisição do co-processador. Em adição, o processador principal usa esse registrador para abortar a execução de uma instrução.

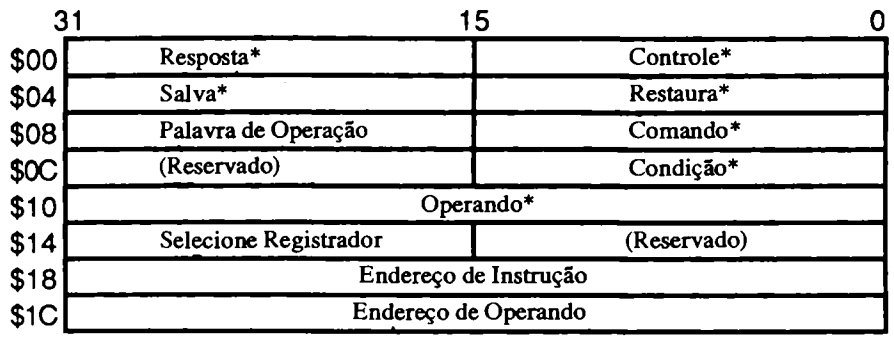


Figura 6.10 Mapa do conjunto de registradores de interface de co-processador.

Registrador de salvamento – um registrador de escrita e leitura de 16 bits. O processador central lê esse registrador para iniciar uma instrução de cpSAVE. O co-processador retorna então ao estado e a informação de formato de quadro de estado para o processador principal através desse registrador.

Registrador de rearmazenamento – um registrador de escrita e leitura de 16 bits. O processador central escreve uma palavra de formato de co-processador nesse registrador para iniciar uma instrução de cpRESTORE. O co-processador então retorna a palavra de formato para o processador principal através desse registrador.

Registrador de palavra de operação – um registrador somente para escrita de 16 bits. O processador principal escreve a palavra de operação F-linha nesse registrador como resposta à requisição de operação de transferência do co-processador.

Registrador de comando – um registrador somente para escrita de 16 bits pelo qual o processador principal inicia uma instrução na categoria de instrução geral (discutida futuramente).

Registrador de condição – registrador somente para escrita de 16 bits pelo qual o processador principal inicia uma instrução da categoria de condicional de co-processador.

Registrador selecionador de registrador – um registrador somente para leitura de 16 bits. O processador principal lê esse registrador para determinar quais registradores serão transferidos quando recebida uma requisição de transferência de registro(s) do co-processador.

Registrador de endereço de instrução – um registrador de leitura/escrita de 32 bits no qual o processador principal transfere o endereço da instrução sendo executada correntemente por uma requisição do co-processador.

Registrador de endereço de operando – um registrador de leitura/escrita de 32 bits. A transferência do endereço do operando é executada através desse registrador dentro da requisição do co-processador.

Há três categorias de instruções de co-processadores: geral, condicional e controle de sistema. A classe geral de instruções é usada para descrever a maioria das instruções de co-processadores e é definida principalmente pelo co-processador. Por exemplo, instruções de soma de ponto flutuante, subtração e multiplicação no co-processador de ponto flutuante MC68881 são exemplos de instruções gerais. As instruções condicionais incluem as de salto (tal como branchs condicionais), e outras instruções condicionais (tal como o set em condição e trap em condição). Em cada caso, o processador principal passa a condição selecionadora para o co-processador para avaliação. O co-processador então indica a condição verdadeira ou falsa para o processador principal, que pode então continuar a execução da instrução. As instruções de controle de sistema incluem duas instruções que permitem chaveamento de tarefas de sistema. Elas são cpSAVE e cpRESTORE. A instrução cpSAVE faz com que o co-processador passe uma palavra de formato e informações de estado interno para o processador principal, que então armazena-os na memória. A instruçao de cpRESTORE faz com que o co-processador carregue seu estado interno com a informação passada pelo processador principal. Ambas as instruções são privilegiadas e assim podem somente ser executadas durante a execução do modo supervisor.

Mais informações sobre a interface de co-processador M68000 podem ser encontradas no *Manual do usuário do MC68020.*

6.10.2 CO-PROCESSADOR DE PONTO FLUTUANTE MC68881

O co-processador de ponto flutuante MC68881, processador em HCMOS, foi

projetado primariamente para ser usado como um co-processador para o MC68020, mas pode também ser operado como um periférico em sistemas não MC68020. Ele suporta completamente o padrão de ponto flutuante P754 da IEEE (versão 10.0). Em adição, ele provê um conjunto completo de funções trigonométricas e logarítmicas não definidas pelo padrão IEEE. Ele executa todos os cálculos internamente com 80 bits de precisão.

A arquitetura do MC68881 aparece como uma extensão lógica da arquitetura do M68000. Quando acoplado ao MC68020 como um co-processador, os registradores do MC68881 podem ser olhados pelo programador como residindo na cápsula do MC68020. A Figura 6.11 mostra o modelo de programação para MC68881. Ele contém oito registradores de ponto flutuante de 80 bits (FP0-FP7), que são análogos aos registradores de dados inteiros (D0-D7) do MC68020, um registrador de controle de 32 bits, um registrador de estado de 32 bits e um registrador de endereço de instrução de 32 bits. O registrador de controle contêm bits de habilitação para cada classe de trap de exceção e bits de modo para a seleção dos modos precisão e arredondamento. O registrador de estado contém códigos de condição de ponto flutuante, bits de quociente e informação de estado de exceção. O registrador de endereço de instrução contém o endereço da última instrução de ponto flutuante executada. Esse registrador é usado para manipulação de exceções.

O MC68881 suporta quatro tipos de dados novos: precisão simples, precisão dupla, precisão estendida e reais de string de decimais compactados, em adição aos três tipos de dados inteiros suportados por todos os processadores M68000 (byte, palavra e palavra longa). A Figura 6.12 mostra os formatos de memória para tipos de dados reais.

O MC68881 suporta cinco classes de instruções de operação: movimentação de dados, operações diádicas, operações monádicas, controle de programa e controle de sistema. As instruções de movimento de dados no MC68881 trabalham essencialmente da mesma forma que as instruções de movimento de dados do MC68020. Elas são usadas para movimentar operandos para, entre e dos registradores do MC68881. Operações diádicas requerem dois operandos e executam funções aritméticas, tais como soma, subtração, multiplicação e divisão. Instruções monádicas provêem funções aritméticas que requerem apenas um operando, tais como sinal, raiz quadrada, negação etc. As instruções de controle de programa afetam o fluxo do programa sob condições do registrador de estado. Essas instruções incluem saltos condicionais no decremento e não-operação, sinalizar condição e testar operando. As operações de controle de sistema são usadas para comunicação com o sistema operacional. As instruções de controle de sistema são: salvar estado, rearmazenar e trap condicionais.

Mais informações sobre MC68881 podem ser encontradas no *Manual de usuário do MC68881*.

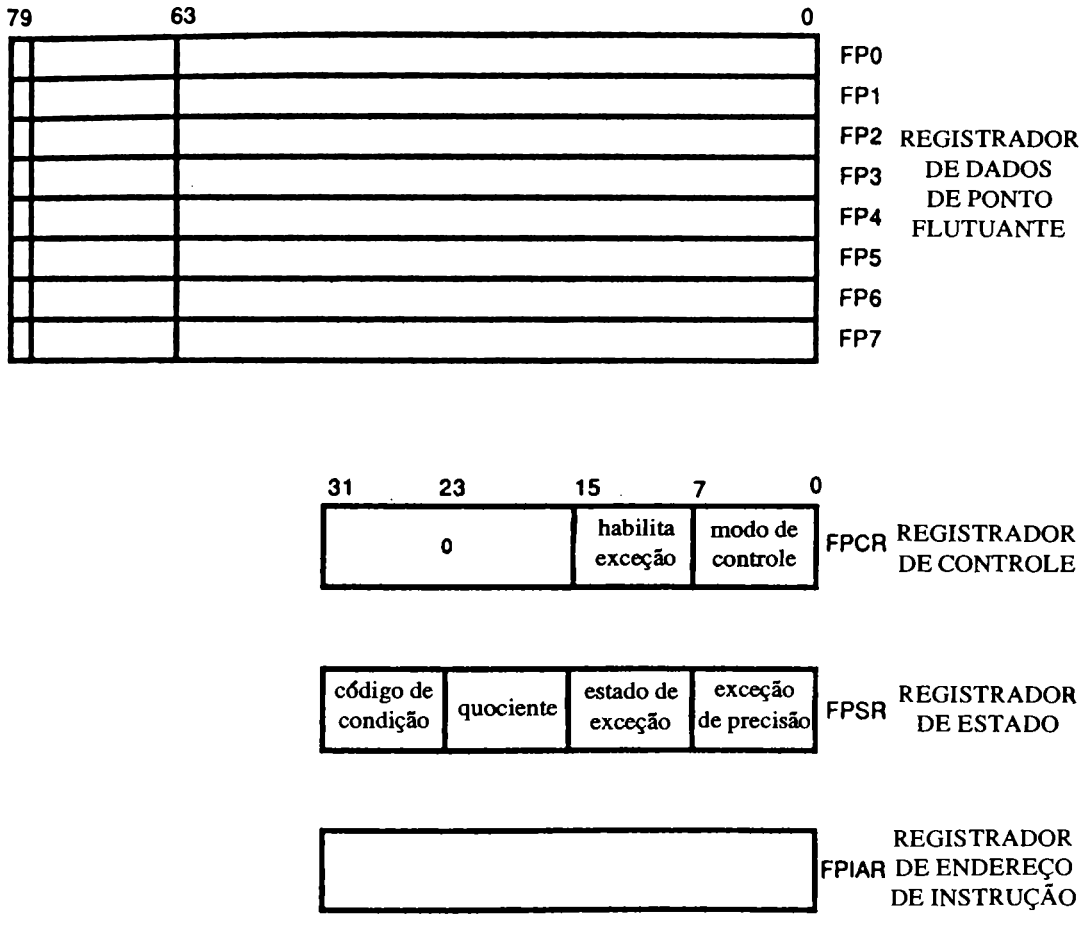


Figura 6.11 Modelo de programação do MC68881.

6.10.3 UNIDADE DE GERENCIAMENTO DE MEMÓRIA PAGINADA MC68851*

A unidade de gerenciamento de memória paginada (PMMU) é também fabricada pelo processo Motorola HCMOS. Ela foi projetada para suportar sistemas de memória virtual paginada por demanda. Ela opera como um co-processador do MC68020, mas pode ser usada com outros processadores. Os parágrafos restantes desta seção discutirão os aspectos de co-processador da PMMU, enquanto a capacidade de gerenciamento de memória será discutida na Seção 6.11.

* N.R. Apesar de grande expectativa, talvez a Motorola não lance comercialmente tão cedo o MC68851.

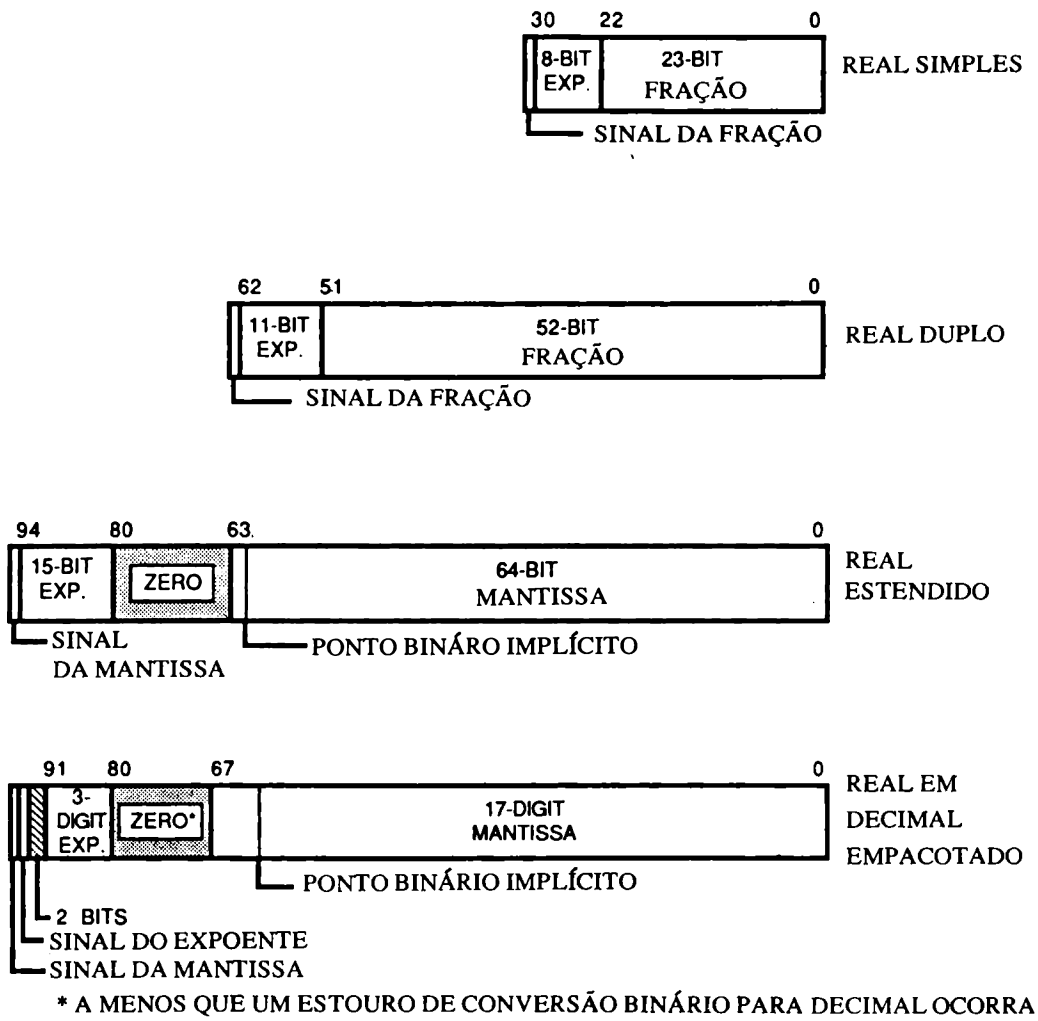


Figura 6.12 Formato de memória para dados do tipo real.

A Figura 6.13 mostra o modelo de programação do MC68851. Os registradores de ponteiro de raiz de CPU (CRP), ponteiro de raiz de DMA (DRP), ponteiro de raiz de supervisor (SRP), controle de transferência (TC), estado de cache (CS), estado (STATUS), nível de acesso corrente (CAL), nível de acesso de validade (VAL), controle de mudança de pilha (SCC) e controle de acesso (AC) controlam tradução e características de proteção da PMMU. Os outros dezesseis registradores, dados de reconhecimento de breakpoint (ponto de parada) (BAD7-BAD0) e controle de reconhecimento de breakpoint (BAC7-BAC0) controlam as funções de breakpoint disponíveis com as instruções BKPT do MC68020.

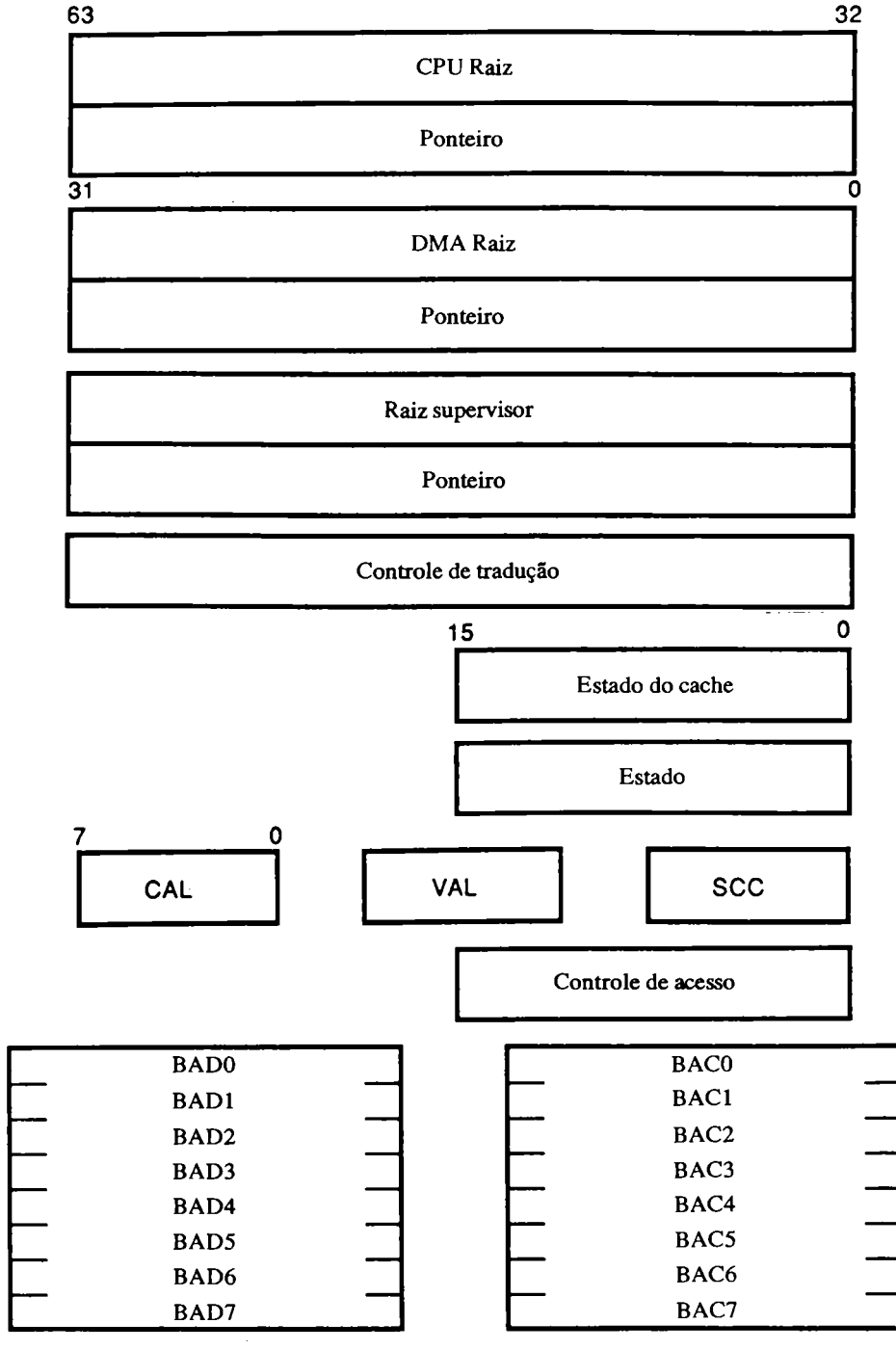


Figura 6.13 Modelo de programação do MC68851.

O registrador CRP tem 64 bits que contêm o ponteiro para a raiz da árvore da tabela de tradução para a tarefa de usuário corrente.

O DRP é um registrador de 64 bits que contém o ponteiro da raiz para a tabela de tradução que é usada quando um mestre do barramento lógico alternado está traduzindo através da PMMU. O SRP é um registrador de 64 bits que contém o ponteiro para a raiz da tabela de tradução a ser usada para tradução de acesso do supervisor. O registrador TC de 32 bits contém bits que configuram o mecanismo de tradução da PMMU. CS é um registrador de 16 bits e somente para leitura que contém bits úteis para a manutenção do cache de dados lógicos. O registrador de estado tem 16 bits e contém bits para indicar erros de barramento, violação do supervisor, violação do nível de acesso, endereços inválidos etc. Os registradores CAL e VAL são ambos de 8 bits; entretanto, somente os três bits mais significativos são implementados. O registrador CAL contém o nível de acesso da rotina sendo executada correntemente, e o VAL contém o nível de acesso do chamador da rotina corrente. O registrador SCC é de 8 bits e determina se a mudança de pilha poderia ocorrer durante uma instrução CALLM do MC68020. O registrador AC de 16 bits controla a informação de acesso para a PMMU (isto é, se os níveis de acesso são habilitados, quantos bits significativos contêm a informação de nível de acesso etc.). Os registradores BAD e CAD têm 16 bits. O registrador BAD contém códigos a serem providos ao processador durante um ciclo de reconhecimento de breakpoint, e um registrador BAC contém as funções de habilitação e contador para a instrução de reconhecimento de breakpoint.

O MC68851 suporta três classes de instruções: carga e armazenamento dos registradores da PMMU (PMOVE), teste de acessos diretos e condicionais (PVALID, PTEST, PLOAD, PFLUSH, PBcc, PDBcc, Scc e PTRAPcc) e funções de controle (PSAVE e PRESTORE). Todas as instruções da PMMU são privilegiadas, exceto a instrução PVALID.

Mais informações sobre o MC68851 podem ser encontradas no *Manual de usuário do MC68851*.

6.11 GERENCIAMENTO DE MEMÓRIA

6.11.1 TÉCNICAS DE GERENCIAMENTO DE MEMÓRIA

O gerenciamento de memória tem basicamente três funções: traduz endereços, provê proteção para o sistema de acesso de usuário e protege contra escrita para algumas

páginas somente para leitura. Sem uma unidade de gerenciamento de memória (MMU), um sistema operacional teria a responsabilidade de manipular tradução de endereços e proteção de acesso por software. Obviamente, isto requeriria uma enorme quantidade de código. Com uma MMU, um sistema operacional precisa somente sinalizar os descritores de tradução. A MMU manipula então as tarefas de tradução e proteção por hardware.

Uma MMU divide o barramento de endereço do sistema, criando endereços lógicos e endereços físicos. O lado lógico é o lado do processador principal, e o barramento físico é usado para acessar a memória principal. A MMU tem duas opções quando traduzindo endereços lógicos para endereços físicos. Na primeira técnica, a MMU soma um deslocamento (offset) definido a um endereço lógico para produzir o endereço físico. No segundo método, chamado técnica de substituição, a MMU procura na tabela de tradução o mapeamento de endereço físico. Tipicamente os métodos de substituição são preferidos porque os endereços físicos podem ser gerados mais rapidamente do que na técnica de soma.

O gerenciamento de memória pode ser implementado em um entre dois modos. A primeira implementação é chamada *segmentada*. Nesse tipo de sistema, um descritor de segmento contém três variáveis: o endereço lógico, o tamanho da janela ou tarefa e o deslocamento. O tamanho define um bloco de memória contínuo (conhecido como *página*) e o deslocamento é usado para gerar o endereço físico. O segundo tipo de implementação é chamado *paginação*. Em um sistema de gerenciamento de memória paginada, são definidas duas variáveis: o endereço lógico e o endereço físico. O tamanho das janelas ou páginas é fixo. Cada sistema tem suas próprias vantagens e desvantagens.

O MC68851 discutido na seção seguinte é uma unidade de gerenciamento de memória paginada.

6.11.2 MC68851

Como colocado na seção anterior, o MC68851 suporta ambientes de memória virtual paginada por demanda. Suporta ainda múltiplos mestres do barramento físico e/ou lógico tão bem quanto caches de dados lógicos e/ou físicos. As funções primárias da PMMU são fornecer tradução de endereços lógicos para físicos, monitorar e reforçar os mecanismos de proteção/privilégio designados pelo sistema operacional e suportar as operações de breakpoint do MC68020. Esses três tópicos serão discutidos nos parágrafos seguintes.

A tarefa de tradução de endereços lógicos para físicos ocupa a maior parte do

tempo da PMMU, e portanto tem sido otimizada em termos de velocidade e intervenção mínima do processador. O MC68851 inicia uma tradução de endereço procurando pelo descritor que descreve a tradução para o endereço lógico apontado pelo cache de tradução no endereço em pastilha (ATC). O ATC é uma associação muito rápida e completa, um cache de 64 entradas que armazena os descritores recentemente usados. Se o descritor não estiver presente no ATC, a PMMU aborta o ciclo, o barramento e torna-se mestre do barramento para "andar" nas tabelas de tradução na memória física. Uma tabela de tradução é uma estrutura de dados hierárquica que contém os descritores de página controlando a tradução de endereços lógicos para físicos. Os registradores de ponteiros de raízes descritos na seção anterior apontam para o topo dessas tabelas de tradução. Quando a MMU encontra o descritor de página correto, ela o carrega na ATC e permite que o mestre do barramento lógico re-tente o ciclo abortado, que poderia ser traduzido corretamente agora.

O mecanismo de proteção do MC68851 provê um exame ciclo por ciclo e reforça os acessos do processo que está sendo executado correntemente. A PMMU suporta oito níveis de privilégio em arranjo hierárquico, que são codificados nos três bits mais significativos do endereço lógico de entrada (LA31-LA29). A PMMU compara esses bits com o valor do registrador CAL (nível de acesso corrente). Se o nível de prioridade do endereço de entrada é menor que o valor do registrador CAL, então o ciclo de barramento está requisitando um privilégio maior que o permitido. A PMMU terminará esse acesso com uma falta (erro). Em adição, a MMU suporta as instruções de chamada e retorno modular do MC68020 (CALLM/RTM). É incluído um mecanismo de alteração de níveis de privilégio durante operação modular.

O MC68851 suporta breakpoints para o MC68020 e outros processadores com a sua capacidade de reconhecimento de breakpoint. Quando o MC68020 encontra uma instrução de breakpoint, ele executa um ciclo de reconhecimento de breakpoint no espaço da CPU. Ele lê uma palavra de um endereço, que é determinado pelo número de breakpoints (especificado na instrução). A PMMU decodifica esse endereço e aloca um código de substituição no barramento de dados ou envia um erro de barramento para indicar uma exceção de instrução ilegal. Os registradores BAD (dado de reconhecimento de breakpoint) contêm os códigos de substituição.

Para mais informações sobre capacidade de gerenciamento de memória do MC68851, reporte-se ao *Manual de usuário do MC68851*.

6.12 SUPORTE A DESENVOLVIMENTO DE SISTEMAS

Os próximos parágrafos descrevem o suporte a sistemas de desenvolvimento disponível para o MC68020.

6.12.1. BENCHMARK 20

O sistema Benchmark 20 provê ao usuário principiante do MC68020 uma ferramenta de avaliação do processador e desenvolvimento de código e depuração para iniciantes. Ele também permite ao usuário executar testes de avaliação no MC68020 via seus utilitários de temporização.

O Benchmark 20, que é baseado em módulo VERSA, suporta assemblers residentes no Motorola nos sistemas EXORmacs e VME/10. Ele consiste em dois quadros VERSAmodule (que também são disponíveis separadamente): o VM04 e o VM13-1. VM04 consiste no MC68020, o MC68881 e a MMB (um MC68461 com um cache de tradução de endereço); ele também tem duas interfaces VERSAbus e RAMbus, um módulo temporizador programável, duas portas de E/S seriais e dois ROM de soquete. A RAMbus é um barramento de extensão de dados/endereço multiplexado assíncrono de 32 bits para ser usado com o VERSAbus. Um subsistema baseado em RAMbus pode transferir dados sem esperar por degradação de desempenho do barramento primário do sistema. O modulo RAM dinâmica VM13-1 tem 1024 kbytes com check de paridade.Como o VM04, ele suporta circuito de check de erro e é porta dual.

O pacote de software disponível com o Benchmark 20 é chamado 020bug. O 020bug é um monitor de depuração com sistema residente em EPROM que suporta assembly e disassembly das instruções do MC68020 e MC68881. Em adição, ele contém utilitário de temporização para execução de programas de tempo.

6.12.2 EMULADOR HDS 400

A estação de desenvolvimento de microprocessador HDS 400 provê emulação em tempo real do MC68020, como também de outros membros da família de processadores M68000. Ele é compatível com o analisador de estado de barramento em tempo real e pode ser suportado pelos mestres EXORmacs, VME/10 ou VAX. Ele suporta todas as características do MC68020, incluindo breakpoint, cache em pastilha e dimensionamento de barramento dinâmico. Em adição, o usuário não tem restrição de acesso aos 4 gigabytes de endereços.

6.12.3 OUTRAS FERRAMENTAS DE DESENVOLVIMENTO

Em adição ao VM04/VM13 mencionados acima, a Motorola também oferece VMEbus compatível com o MC68020. O VME130 roda a 12,5 MHz e contém soquetes para o MC68881 e o MC68851. Contém também soquetes para 16 Kbytes de ROM/EPROM, além de duas portas de E/S serial para multiprotocolos, uma interface de barramento privada MVMX32bus e um módulo de temporização programável. O VME203 é um módulo de memória dinâmica de 1 Mbyte projetado para ser usado com o VME310. Ele é porta-dual, tem uma interface MVMX32bus e provê check de erro e byte de paridade.

6.12.4 SISTEMA 1131

O Sistema 1131 suporta uma combinação de MC68020/MC68881/MC68851 usando a configuração VMEboard. O sistema inclui o quadro VME131, que contém o MC68020, o MC68881 e o MC68851, floppy de 1 Mbyte e um disco rígido de 15 Mbyte, tudo montado em um chassi rackmount VME. Um completo monitor de depuração é incluído no quadro de ROM. O sistema UNIX V/68 pode ser obtido para o quadro a um custo baixo.

6.13 SUPORTE PARA SOFTWARE

Existe uma grande escala de suporte para sofware para o MC68020. Os parágrafos seguintes descrevem os cross assemblers e os cross compiladores disponíveis para o MC68020. Também descrevem o UNIX e outros suportes a software.

Cross Assemblers – Cross assemblers são disponíveis para o MC68020 por ambas estações de desenvolvimento VME/10 e EXORmacs. Esses cross assemblers suportam ambos sistemas operacionais SYSTEM V/68 e VERSAdos.

Cross Compiladores – Duas versões de cross compiladores C-compiler estão disponíveis pela Motorola. A primeira versão roda com o sistema operacional SYSTEM V/68 em ambas as estações de desenvolvimento EXORmacs e VME/10. A segunda versão roda no VAX/780 com o sistema operacional UNIX™ SYSTEM V (1 e 2). Esse pacote inclui o assembler do SYSTEM V/68 e o C-compiler. Um compilador FORTRAN 77 é também disponível com o UNIX™ SYSTEM V (2).

Suporte UNIX™ – O suporte UNIX™ para o MC68020 é disponível pela Motorola e

outras casas de software. O UNIX V™ (AT&T versão 2 e 2+) é oferecido pela Motorola para suportar ambos VM04 e VME130. Esses pacotes de software também incluem FORTRAN 77. Suporte adicional UNIX™ para o MC68020 é oferecido pela UNISOFT, novamente para ambas VM04 e VME130.

Terceiros suportes – A família M68000 de microprocessadores tem um largo e diverso suprimento de software de terceiros fornecedores. As aplicações vão de serviços e indústria até sistemas operacionais e compiladores. A Motorola publicou um catálogo de software que contém uma lista completa dos vendedores de software e seus produtos.

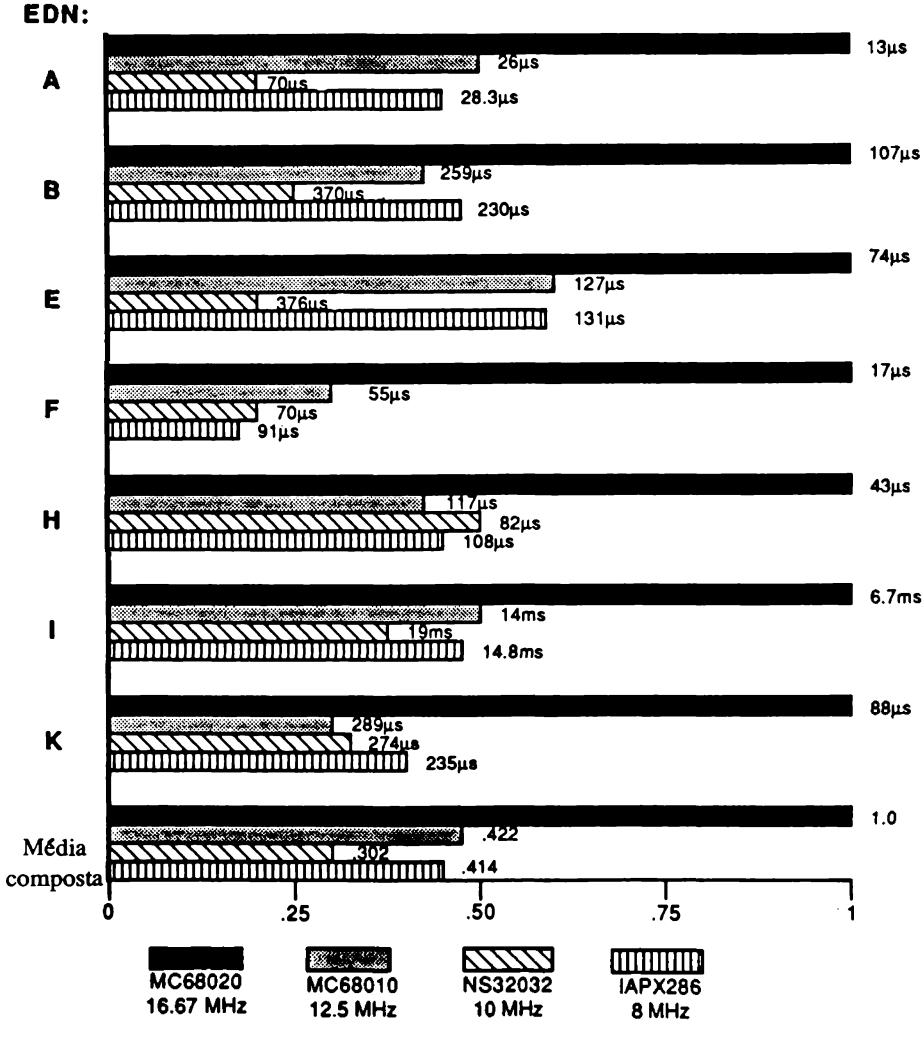


Figura 6.14 Testes de desempenho EDN.

6.14 DESEMPENHO

Os projetistas do MC68020 tinham um objetivo em mente: alto desempenho. O fato desse objetivo ter sido alcançado é evidenciado com poucos números. Primeiro, os melhoramentos feitos no MC68020 sobre o M68000 de 8 MHz será notado.

O MC68020 foi projetado para rodar a 16 MHz dando duas vezes de otimização em relação ao MC68000. Ele tem um barramento de dados de 32 bits, enquanto o MC68000 tem o barramento de dados de 16 bits, dando um melhoramento de 1,3 vezes. Note que isto não é um melhoramento de duas vezes, porque nem todos os 32 bits do barramento são usados sempre. Um cache de instruções foi adicionado para dar uma otimização de 1,25 vezes. Isto leva em consideração um tempo de execução de ciclo de barramento mais rápido, otimização de banda de barramento etc. Finalmente, novas instruções, modos de endereçamento e uma ALU de 32 bits (unidade lógica aritmética) foram implementados com 1,25 vezes de otimização. O resultado de todos os melhoramentos é um processador que tem um desempenho 4,06 mais alto que o MC68000 de 8 MHz.

A Figura 6.14 mostra os tempos de execução do MC68020 comparados com outros processadores por vários testes de avaliação.

Em adição ao desempenho, os projetistas do MC68020 tinham outros objetivos em mente. Mais importante, o MC68020 tinha de ser compatível com outros membros da família M68000. Portanto software existentes em uso não precisariam ser reescritos para rodar no MC68020. Eles também sentiram a necessidade de somar um conjunto de instruções significantes ao processador. Isto fez o MC68020 útil para mais tipos de aplicações, tal como robótica, gráficos e estações de trabalho de engenharia. A evidência disto pode ser encontrada no número de variações de sistemas baseados no MC68020 já introduzidos pelas várias companhias internacionais.

REFERÊNCIAS

MC68020 32-bits Microprocessors User's Manual (2 ed.), Prentice-Hall, Inc., Englewood Cliffs, NJ, 1985.

MC68881 Floating Point Coprocessor User's Manual, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1985.

MC68851 Paged Memory Management Unit, Motorola, Inc.

Motorola Microprocessor Software Catalog (4 ed.), Motorola, Inc., 1985.

"The Motorola MC68020", Macgregor, Doug, Mothersole, Dave e Moyer, Bill. IEEE, agosto, 1984.

CAPÍTULO **7**

A CPU ZILOG Z80000

Bradly K. Fawcett
Zilog Inc., USA

7.1 INTRODUÇÃO

Continuando a tradição da Zilog em projetos de componentes de microprocessadores estado de arte, o microprocessador de 32 bits Z80000 traz o desempenho de superminicomputadores e computadores de grande porte para o reino dos sistemas baseados em microprocessadores. Além da compatibilidade de software e hardware com a família Z8000, a CPU Z80000 caracteriza uma alta capacidade, uma arquitetura de 32 bits que suporta diretamente sistemas operacionais e linguagens de alto nível. Sua interface de hardware flexível provê conexões com uma grande variedade de configurações de sistemas.

A CPU Z80000 tem acesso a todos os endereços e dados de 32 bits, e pode endereçar diretamente mais de 4 gigabytes de memória virtual. A capacidade é melhorada por instruções canalizadas internas de seis estágios, cache de memória em pastilha, suporte para transações de memória modo-burst, gerenciamento de demanda de páginas de memória, uma interface para co-processador, altas velocidades de relógio – um conjunto de instruções rico.

Várias aplicações Z80000 envolverão o uso de sistemas operacionais multitarefa e linguagens de alto nível. As características de arquitetura que suportam a implementação de tais sistemas incluem sistemas separados e modos de operação normal, gerenciamento de memória em pastilha, manipulação sofisticada de interrupção e trap, um grande arquivo de registradores de propósito geral e um conjunto de instruções grande e poderoso. São usados nove operandos de modos de endereçamento dentro das instruções para acessar numerosos tipos de dados, incluindo bits, campos de bits, inteiros sinalizados e não sinalizados, valores lógicos, strings e dígitos BCD. O conjunto de instruções é altamente regu-

lar para combinações de instruções, modos de endereçamento e tipos de dados. A compilação de linguagens de alto nível é suportada por instruções para enlaces de procedimentos, cálculos indexados de vetores, conversão de tipos de dados e verificação de limites de tamanho; outras operações fornecem funções de sistemas operacionais, tal como chamadas a sistema, teste de semáforo e gerenciamento de memória.

O barramento externo da CPU Z80000 provê, para fácil conexão, uma grande variedade de ambientes de sistemas, aumentando a ordem de requisição de custo/desem-

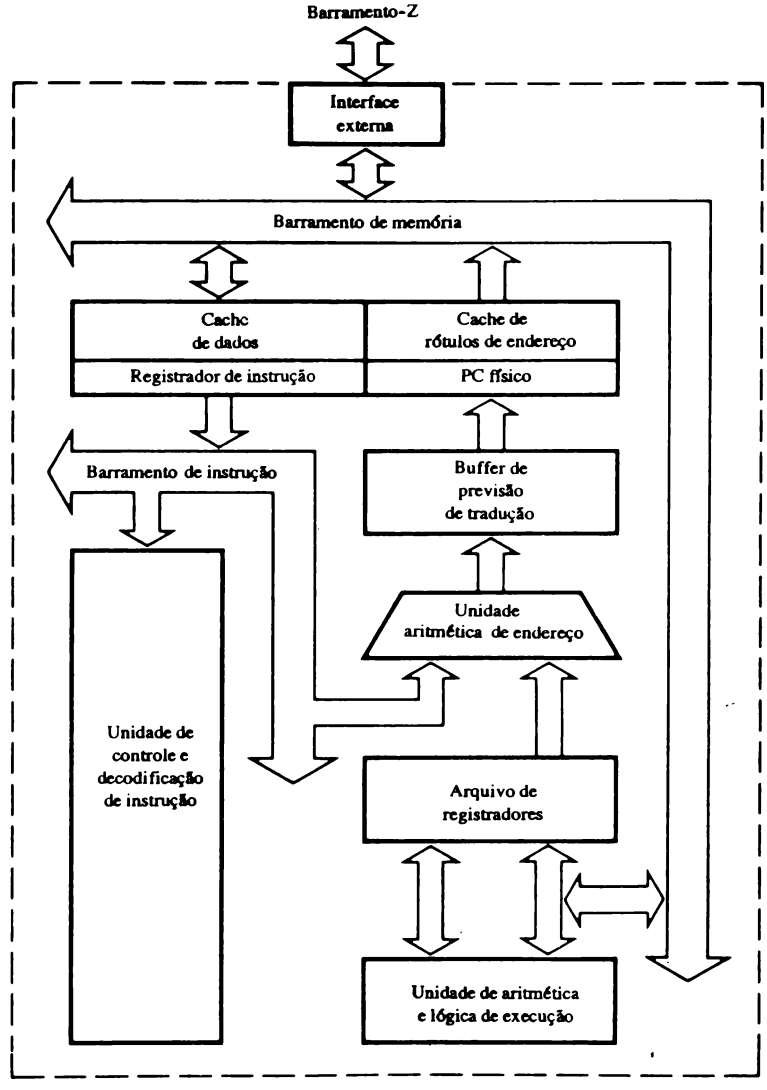


Figura 7.1 Diagrama de blocos funcional do Z80000.

penho. O projetista de sistemas pode, controlando por programa, configurar o período de relógio para barramento, a quantidade de passagem de dados e o tempo de acesso para transações no barramento. Opcionalmente, as transações de memória modo-burst podem ser usadas para aumentar a banda de memória. São suportados diretamente quatro tipos de configuração de multiprocessamento: co-processadores, processadores escravos, múltiplas CPU precisamente emparelhadas, e múltiplas CPU imprecisamente emparelhadas. A estrutura do barramento é compatível com o barramento-Z (Z-bus) de toda a família de processadores, co-processadores e periféricos.

A Figura 7.1 mostra um diagrama de bloco funcional da organização interna da CPU Z80000. A unidade de interface externa controla as transações externas do barramento; a interface externa Z-bus consiste em um barramento multiplexado por tempo para dados e endereços e seus estados associados e sinais de controle. Uma memória cache retém a cópia da maioria das localizações das últimas instruções e dos últimos dados acessados. A unidade aritmética de endereços realiza todos os cálculos de endereços efetivos e os transmite pelo buffer de saída (Translation Lookside Buffer – TLB), que transforma o endereço lógico em endereço físico. Os arquivos de registradores contêm 60 registradores de 32 bits de propósito geral e vários registradores de controle de propósito especial. A unidade de execuções aritméticas e lógicas (ALU) calcula os resultados da execução da instrução; essa unidade tem duas passagens para o arquivo de registradores, permitindo que dois operandos sejam lidos simultaneamente. O decodificador de instruções e unidade de controle decodifica as instruções e controla as operações de outras unidades funcionais. Todas as unidades funcionais e passagens de dados são de 32 bits. A operação da CPU é altamente canalizada, com as unidades funcionais operando em paralelo.

O projeto da CPU Z80000 é baseado em transistores de 2 μm fabricados em um processo especial que caracteriza-se por transistores intrínsecos de fabricação NMOS, com transistores de alto valor de polissilício, e múltiplos níveis de interconexão. Com esse processo, é possível se ter velocidades de relógio de CPU de mais de 25 MHz.

7.2 ESPAÇOS DE ENDEREÇO

A CPU Z80000 pode manipular dados tipo bits, bytes (8 bits), palavras (16 bits), palavras longas (32 bits) e palavras quádruplas (64 bits). Os dados podem ser alocados dentro de qualquer dos vários endereços lógicos, incluindo os quatro intervalos de endereços de memória, o intervalo de endereço de E/S e o arquivo de registradores da CPU.

Para referência à memória e E/S, a CPU Z80000 transforma o endereço lógico especificado pelo programa em endereço físico, que é enviado pelo barramento de dado/endereço e indicado à memória real ou dispositivo de E/S. Os endereços lógicos (isto

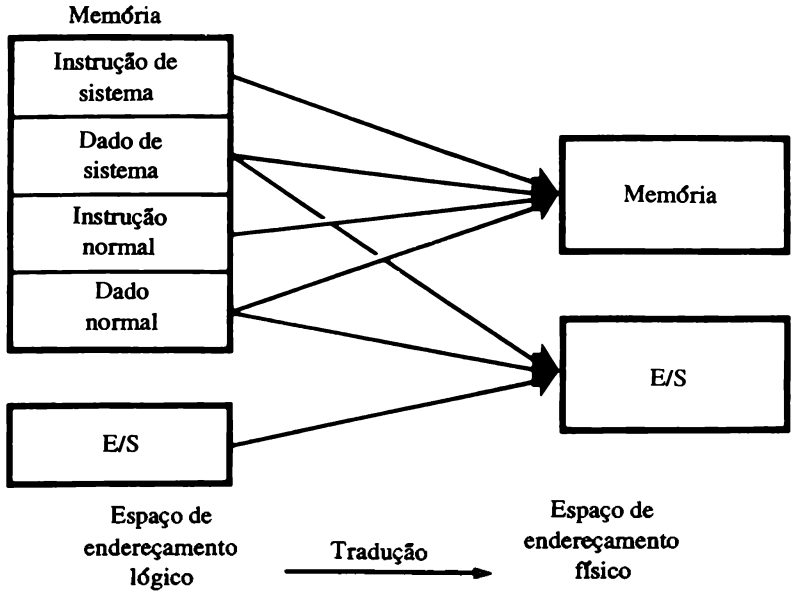


Figura 7.2 Espaços de endereços.

é, os endereços manipulados pelo programa) estão em um dos quatro intervalos de endereço de memória lógica ou no intervalo de endereço de E/S lógico. Os endereços físicos estão no intervalo de memória física ou no intervalo de E/S físico (Figura 7.2).

Os quatro intervalos de endereço de memória lógica são o intervalo de instruções do sistema, intervalo de dados do sistema, intervalo de instrução normal e intervalo de dados normais. A unidade de gerenciamento de memória em pastilha pode controlar independentemente cada um dos intervalos de memória lógica. Quando em modo sistema, as referências de memória acessam um dos intervalos de endereços do sistema; em modo normal, as referências à memória acessam um dos intervalos de endereços normais. O intervalo de instruções é usado para leitura de instruções (fetch), leitura de operandos imediatos e acesso a dados usando o endereço relativo ou os modos de endereçamento indexados relativos; o intervalo de endereço de dados é usado para referência a dados usando qualquer outro modo de endereçamento.

Os endereços de memória lógicos são sempre de 32 bits. Qualquer um dos três diferentes modos de representação de endereço de memória pode ser usado: compactado, segmentado ou linear (Figura 7.3).

No modo compactado, são manipulados pelo programa endereços de memória de 16 bits. Os cálculos de endereços efetivos usando-se endereços compactados envolvem

todos os 16 bits. O modo compactado pode ser usado para aplicações que necessitem menos de 64 kbytes de código e que acessem menos de 64 kbytes de dados. O modo compactado é mais eficiente e consome menos espaço de programa que o modo linear ou segmentado, desde que endereçamentos da memória no programa sejam de uma palavra em vez de duas. No modo compactado os 16 bits mais significativos de cada endereço de memória lógica de 32 bits são os 16 bits menos significativos do contador de programa.

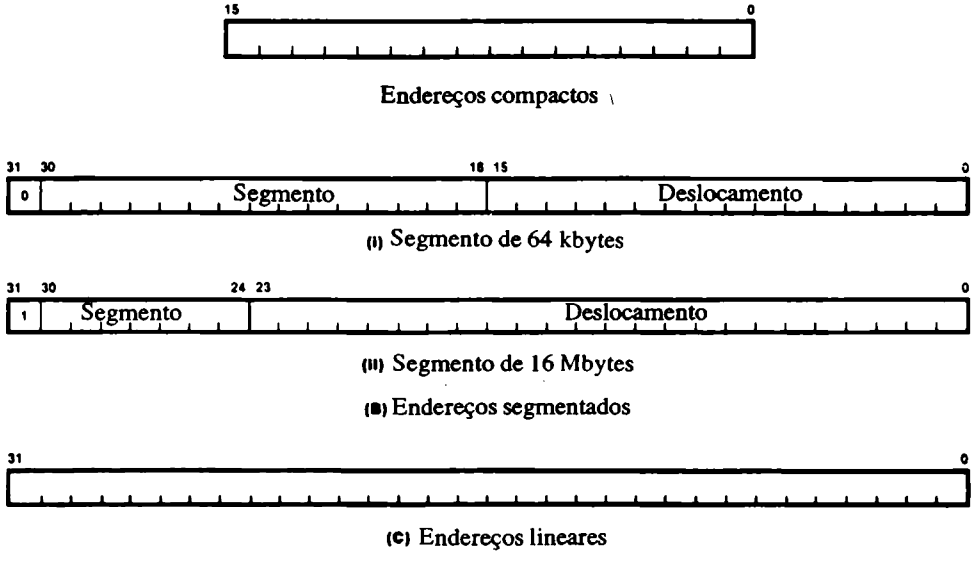


Figura 7.3 Representações de endereços de memória.

Em modo segmentado, os intervalos de endereços de memória lógicos são divididos em dois segmentos diferentes de memória. A metade mais baixa de cada intervalo de endereço é dividida em 32768 segmentos de mais de 64 kbytes cada e a metade mais alta é dividida em 128 segmentos de mais de 16 Mbytes cada. Cada segmento é um string contínuo de bytes em deslocamentos (offset) contínuos. Cada endereço de memória é composto de um número distinto de segmentos e do endereço do offset (Figura 7.3). Os cálculos de endereço efetivo envolvem somente a parte de deslocamento (offset) do endereço; o número de segmentos não é afetado. Segmentação é uma técnica de organização de memória comum usada em minis e computadores de grande porte; muitas aplicações se beneficiam da estrutura lógica de segmentação pela alocação individual dos módulos do programa, pilhas ou estrutura de dados em segmentos separados.

Em modo linear, os cálculos de endereço efetivo envolvem todos os 32 bits de endereço; o intervalo de endereço de 4 gigabytes é uniforme e não estruturado. Dessa forma, todo o intervalo de endereço será um vetor contínuo de bytes em endereços conse-

cutivos. Algumas aplicações são beneficiadas pela flexibilidade de endereçamento linear, onde os objetos podem ser associados a qualquer local arbitrário na memória.

A memória em sistemas Z80000 é endereçável por byte – isto é, cada byte de memória tem seu próprio endereço. Quando se armazena na memória uma palavra ou uma palavra longa de dado, o dado é armazenado em locais consecutivos na memória, iniciando-se com o byte mais significativo (Figura 7.4). Palavras e palavras longas na memória são endereçadas usando-se o endereço mais baixo de qualquer byte no dado (isto é, o endereço do byte mais significativo). Os dados de uma palavra ou palavra longa na memória podem começar em qualquer endereço; entretanto, o desempenho é melhor quando as palavras começam em endereço par e as palavras longas começam em endereços múltiplos de quatro. As instruções de palavras devem começar em endereços pares.

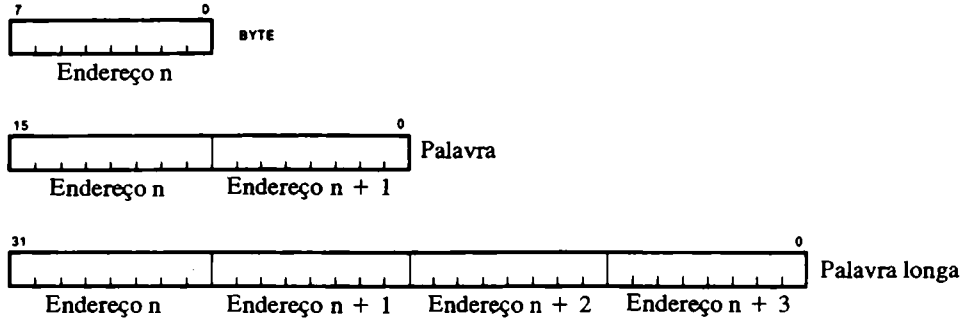


Figura 7.4 Armazenagem de dados em memória.

Os endereços de E/S lógicos são de 32 bits, mas somente os bits menos significativos são manipulados pelo programa; a CPU força os 16 bits mais significativos a serem todos zero. As portas de E/S podem ser byte, palavra ou palavra longa.

Os endereços físicos estão no intervalo físico da memória ou no intervalo físico de E/S. Os dois intervalos de endereços físicos são distinguidos por sinais de estado no barramento diferentes e tempos de transações diferentes. Um endereço no intervalo de endereços de E/S lógico mapea para o endereço idêntico no intervalo de endereços de E/S físico. O dispositivo de gerenciamento de memória em pastilha mapeia endereços lógicos de memória para o intervalo de memória física ou o intervalo de E/S físico (para dispositivos de E/S com memória mapeada).

O armazenamento de dados nos registradores da CPU resulta em instruções menores e execução mais rápida do que com instruções que acessam a memória ou os intervalos de endereços de E/S. A CPU Z80000 é um processador orientado a registrador que contém 60 registradores de 32 bits de propósito geral, um contador de programa de 32 bits

(PC), uma palavra de controle e indicadores de 16 bits (FCW) e outros nove registradores de controle de proposta especial.

O arquivo de registradores de propósito geral contém 60 registradores de 32 bits, para um total de 64 bytes de armazenamento (Figura 7.5). Os formatos de dados para registradores, desde bytes até palavra quádrupla, são criados dividindo-se e agrupando-se os registradores de palavra longa. Por exemplo, o registrador de palavra longa RR0 é subdividido em registrador de palavra R0 e R1, o registrador de palavra é subdividido em registrador de bytes RH0 e RL0, e assim por diante. Dois registradores de palavra longa são aglutinados para formar um registrador de palavra quádruplo, os registradores RR0 e RR2 juntos formam o registrador de palavra quádruplo. Desse modo, o programador Z80000 pode endereçar separadamente 16 registradores bytes, 16 registradores de palavra, 16 registradores de palavra longa e 8 registradores de palavra quad. O resultado é um arquivo de registrador que provê a máxima flexibilidade para manipulação de tipos de dados diferentes. O programador pode especificar o tamanho apropriado de registrador para cada dado sem sacrificar espaço de registradores adicional.

Esses registradores são de propósito geral por natureza e podem ser usados para armazenar dados e endereços. Registradores de bytes podem ser usados como acumuladores para operações de 8 bits. Registradores de palavras podem ser usados como acumuladores para operações de 16 bits, como indexadores de registradores ou como ponteiros para memória (no modo compactado). Os registradores de palavra longa podem ser usados

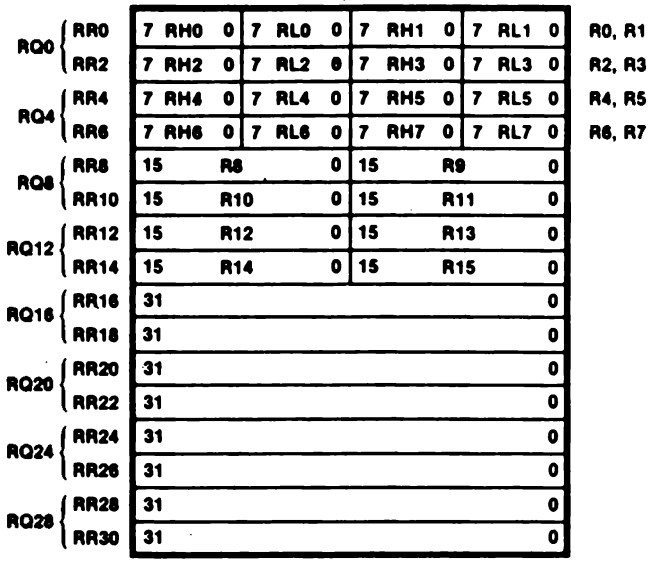


Figura 7.5 Arquivo de registradores de uso geral.

como acumuladores para operações de 32 bits, como registradores indexados ou como ponteiros a memória (nos modos segmentado e linear). Os registradores de palavra quad podem ser usados como acumuladores para resultados de 32 bits de multiplicações, divisão ou instruções de sinal estendido. Desde que todos os registradores são de propósito geral, uma forma de uso particular de um registrador pode variar no curso do programa, dando ao programador uma grande flexibilidade. Essa arquitetura permite que o programa sobrecarregue o uso da arquitetura de um registrador dedicado ou subentendido, ao qual o conteúdo deverá ser salvo e rearmazenado sempre que a necessidade de registradores de um tipo particular exceda o número de registradores daquele tipo no processador.

Dois dos registradores de propósito geral são dedicados para o ponteiro a pilha e ponteiro ao quadro usado pelas instruções Call, Enter, Exit e Return. O ponteiro usado depende do modo de representação de endereçamento de memória corrente usado. No modo compactado, o R15 é o ponteiro da pilha e o R14 é o ponteiro de quadro, enquanto em modo segmentado ou linear o RR14 é o ponteiro da pilha e o RR12 é o ponteiro de quadro. Estes são os ponteiros de pilhas separados para os modos de operação sistema e normal.

Em adição ao arquivo de registradores de propósito geral, a CPU Z80000 também contém dois registradores de estado do programa e nove registradores de controle de propósito especial. Os registradores de estados do programa são o contador de programa (PC), que indica o endereço da próxima instrução a ser executada, e a palavra de estado e controle (FCW), que sinaliza os bits de controle que determinam os modos de operação da CPU e os estados dos indicadores da última operação da ALU. Os registradores de propósito especial são usados para gerenciamento de memória, configuração do sistema e outras funções de controle da CPU.

A palavra de 16 bits de estado e controle contém informações de controle e estados. O byte menos significativo contém seis indicadores (carry, zero, sinal, paridade/overflow, ajuste decimal e half-carry) e a máscara de overflow de inteiros. Os indicadores refletem o resultado da última operação, e são modificados e usados por várias instruções. Através do controle do programa o bit habilitador de overflow de inteiro é usado para habilitar e desabilitar a detenção de overflow de inteiros. O byte mais significativo da FCW contém oito bits de controle. Dois bits selecionam o modo de representação de endereçamento de memória corrente (compactado, segmentado, linear). O bit de modo de arquitetura de processador estendido é usado para informar a CPU se os co-processadores estão presentes no sistema. Bits de habilitação distintos são providos para entradas de interrupção vetorizadas e não vetorizadas. Bits de trace e trace pendente são usados durante o processo de depuração passo a passo. O bit sistema/norma determina o modo de operação corrente da CPU.

A habilidade da CPU para operar em modos sistema e normal separadamente facilita a implementação de sistemas operacionais protegidos. O modo operação determina qual instrução pode ser executada e qual ponteiro de pilha será usado. Todas as instruções podem ser executadas no modo sistema; no modo normal, as instruções que afetam o hardware diretamente, tal como instruções de E/S, não podem ser executadas (as instruções que não podem ser executadas no modo normal são chamadas *instruções privilegiadas*). O software do sistema operacional poderia rodar em sistema normal. Os mecanismos de gerenciamento de memória permitem que programas em modo sistema acessem áreas de memória protegidas do uso do modo normal. Dessa forma, o sistema operacional e o hardware por si são protegidos automaticamente de operações do modo normal. Outras proteções são fornecidas com ponteiros a pilha separados para os modo normal e sistema. Programas de aplicação de modo normal podem usar traps para requisitar serviços do sistema operacional.

Os nove registradores de controle de propósito especial da CPU Z80000 estão descritos resumidamente abaixo. São de 32 bits cada um e, desde que eles controlam a configuração do sistema, podem ser acessados durante operações no modo sistema.

- *Ponteiro a área de estado de programa* (PSAP – Program Status Area Pointer). Contém o endereço de memória física inicial da área de estados de programa. A área de estados de programa é a tabela de memória que contém os valores carregados nos registradores de estados de programa durante a interrupção e o processamento de trap.

- *Ponteiro normal de pilha* (NSP – Normal Stack Pointer).O ponteiro de pilha usado enquanto está no modo normal (isto é, o modo normal RR14 ou R15). Este registro permite que o ponteiro normal de pilha seja acessado enquanto está no modo sistema.

- *Descritor da tabela de tradução de instruções de sistema* (SITTD – System Instruction Translation Table Descriptor). Contém o endereço de memória física da tabela de tradução para leitura (fetch) de instruções do modo sistema e outras informações de controle de gerenciamento de memória.

- *Descritor da tabela de tradução de dados de sistema* (SDTTD – System Data Translation Table Descriptor). Contém o endereço de memória física da tabela de tradução para leitura de dados no modo sistema e outras informações de controle de gerenciamento de memória.

- *Descritor da tabela de tradução de instruções do modo normal* (NITTD – Normal Instruction Translation Table Descriptor). Contém o endereço de memória física da tabela de tradução para leitura de instruções do modo normal e outras informações de controle de gerenciamento de memória.

• *Descritor da tabela de tradução de dados normal* (NDTTD – Normal Data Translation Table Descriptor). Contém o endereço de memória física da tabela de tradução para leitura de dados no modo normal e outras informações de controle de gerenciamento de memória.

• *Ponteiro de overflow de pilha* (OSP – Overflow Stack Pointer). Contém o endereço físico da área de overflow da pilha que é usado quando um erro de tradução ocorre durante a interrupção ou processamento de trap.

• *Registrador de interface de hardware* (HICR – Hardware Interface Control Register). Controla a configuração da interface do barramento externo, incluindo a velocidade do barramento, quantidade de passagem de dados e estados de espera automáticos.

• *Palavra longa para controle de configuração do sistema* (SCCL – System Configuration Control Long Word). Contém controles para a unidade de gerenciamento de memória, mecanismos de cache e lógica de processamento de exceções.

7.3 GERENCIAMENTO DE MEMÓRIA

A CPU Z80000 contém um mecanismo de gerenciamento de memória em pastilha que provê tradução de endereços lógicos para físicos e proteção a acesso à memória. Mapeando os endereços de memória lógica para endereços físicos, uma tarefa programada pode ser alocada em qualquer lugar da memória física. Desse modo, múltiplas tarefas programadas que usam o mesmo endereço lógico podem ser mapeadas em áreas diferentes da memória física. Reciprocamente, compartilhamento de memória pode ser implementado mapeando-se endereços lógicos diferentes nos mesmos endereços físicos. O dispositivo de gerenciamento de memória também limita os tipos de acesso que podem ser feitos a uma área física da memória, provendo um mecanismo de proteção para garantir a segurança de códigos ou dados sensíveis, tal como o código do sistema operacional. O gerenciamento de memória da CPU Z80000 suporta a demanda de ambientes de memória paginada virtualmente, onde o uso de endereços lógicos excede a quantidade de memória física disponível.

A lógica de gerenciamento de memória pode ser habilitada ou desabilitada independentemente para os intervalos de endereços de memória do modo normal e sistema via a programação do registrador de palavra longa de controle de configuração do sistema. Quando desabilitado, o endereço físico é idêntico ao endereço lógico e todos os acessos são permitidos.

O esquema de tradução do gerenciamento de memória divide logicamente o in-

tervalo de endereços lógicos em páginas e o intervalo de endereços físicos em quadros; ambos, página e quadros, são sempre de 1 kbyte. O processo de tradução envolve mapeamento de uma página, identificada pelos 22 bits mais significativos do endereço lógico, para um quadro específico, identificado pelos 22 bits mais significativos do endereço físico inicial daquele quadro. Os dez bits menos significativos, que indicam uma posição específica dentro de uma página ou quadro, são idênticos no endereço lógico e físico. A CPU tem um buffer de saída de tradução (TLB) que armazena informações de tradução para os dezesseis quadros mais recentemente referenciados em um buffer de memória inteiramente associativo. Para cada referência à memória, o endereço lógico é comparado ao rótulo de endereço na TLB. Se for encontrado um correspondente, a entrada da TLB é usada para produzir um endereço físico para aquela referência (Figura 7.6). Quando a informação para aquela página não está na TLB, a CPU automaticamente referencia a tabela de tradução na memória, carregando a nova informação na TLB e repondo a entrada recentemente usada da TLB. Dessa forma, a TLB atua como um buffer que é automaticamente carregado com a informação de tradução mais recentemente usada.

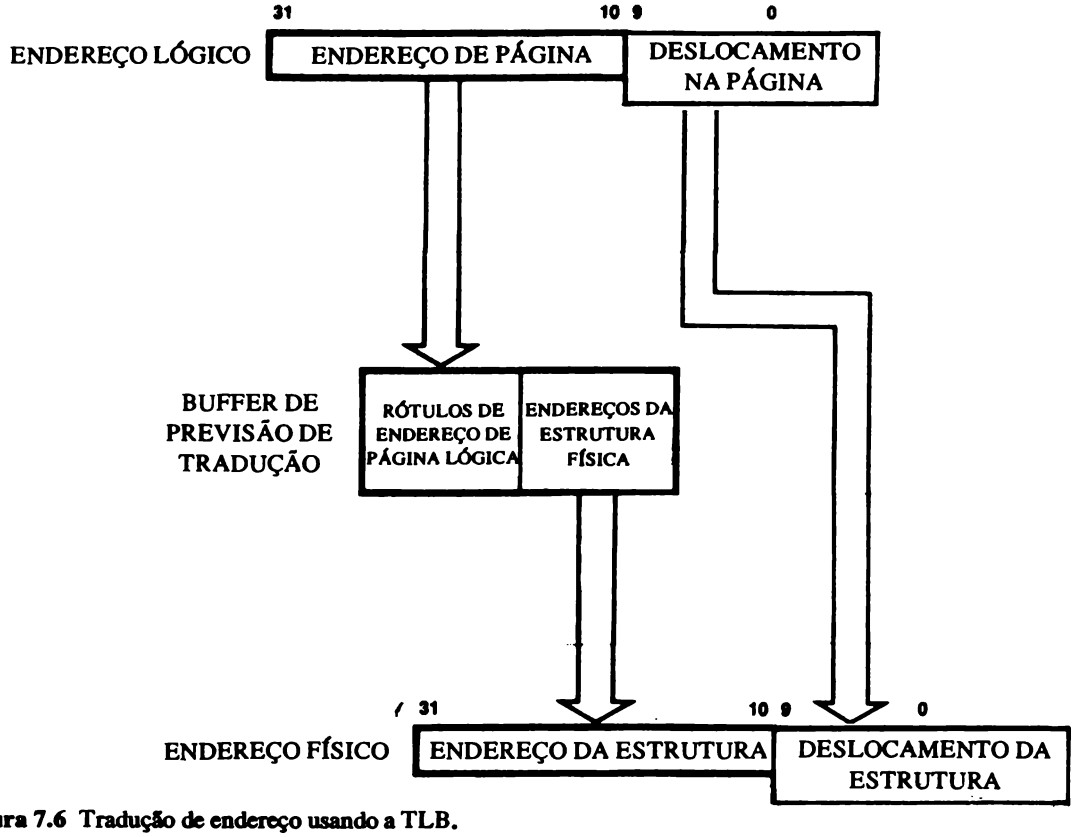


Figura 7.6 Tradução de endereço usando a TLB.

A tradução de endereços e o esquema de proteção empregam três níveis de tabela. O endereço lógico é particionado em um campo de nível 1 de 8 bits, um campo de nível 2 de 8 bits, um campo de número de página de 6 bits, e um campo de offset de página de 10 bits (Figura 7.7). Quando a informação de tradução é carregada na TLB, a CPU lê automaticamente as entradas de três níveis de tabelas, usando os campos no endereço lógico como indicadores das tabelas correspondentes. Os registradores de descrição de tabelas de tradução na CPU referenciam o endereço inicial da tabela de nível 1 para aquele intervalo de endereço. A entrada da tabela de nível 1 contém o endereço inicial da tabela de nível 2; a entrada da tabela de nível 2 contém o endereço inicial para a tabela de páginas; a entrada da tabela de páginas contém o endereço físico inicial para aquele quadro, que é carregado na TLB. Existem várias características opcionais que permitem reduzir o número de níveis e o tamanho das tabelas. Por exemplo, quando o intervalo de endereços não é usado totalmente, os níveis de tabelas podem ser seletivamente pulados. As tabelas de nível 1 podem ser puladas quando um intervalo de endereço de 16 Mbytes é usado; os níveis 1 e 2 de tabelas podem ser pulados para endereços compactados.

Junto ao ponteiro para o próximo nível de tabela, cada entrada na tabela de tradução contém informações de proteção a acessos. Pode-se controlar independentemente três tipos de proteção: execução, leitura e escrita. Pode-se associar separadamente proteções para acessos a modo sistema e normal. A proteção pode ser especificada em qualquer nível de tabela de tradução.

Durante um acesso à memória, se a lógica de gerenciamento de memória da CPU, detecta uma violação da proteção de acesso ou uma entrada de tabela não válida, a instrução que está sendo executada é suspendida e um trap de tradução de endereço é executado. A CPU salva automaticamente o estado dos registradores e memória, para que a instrução possa ser reiniciada depois da eliminação da condição de violação, de uma maneira compatível com as exigências de memória virtual.

7.4 MODOS DE ENDEREÇAMENTO DE OPERANDOS

As instruções do Z80000 podem manipular operando de dados alocados em registradores, memória ou portas de periféricos. São disponíveis nove modos de endereçamento de operandos para especificar-se o endereço do operando dentro de uma instrução. A maioria das instruções pode usar qualquer modo de endereçamento, embora algumas instruções suportem somente um subconjunto dos nove modos.

A CPU pode ser requisitada para realizar cálculos de endereço efetivo do endereço do operando quando acessa operandos na memória. O cálculo de endereço efetivo

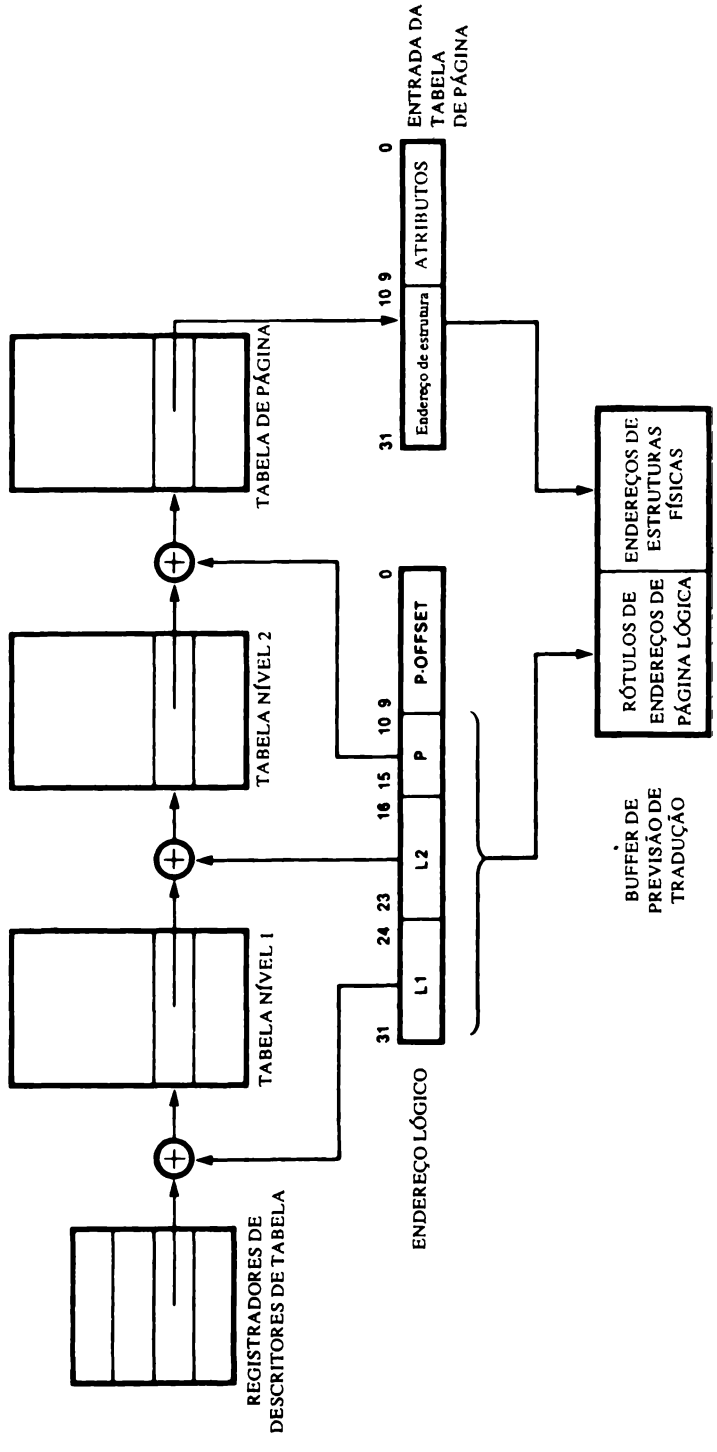


Figura 7.7 Carga da TLB das tabelas em memória.

envolve valores de soma e indexação e/ou valores de deslocamentos a partir de um endereço base. O endereço base é alocado na instrução, ou em um registrador de propósito geral, ou no PC; o valor de indexação é alocado em um registrador de palavra ou palavra longa de propósito geral; o deslocamento é alocado na instrução.

Em modo compacto, os endereços são de 16 bits, e os cálculos de endereços efetivos usam aritmética de 16 bits. O carry e o overflow do bit mais significativo é ignorado. Assim, os endereços circulam com endereço 65535 seguido do endereço 0.

Em modo segmentado, somente a porção de offset do endereço base é usada no cálculo de endereço efetivo. O tamanho do segmento e o número do segmento do endereço efetivo são sempre o mesmo que do endereço base. Os cálculos de endereços efetivos usam aritmética de 16 bits para os segmentos de 64 kbytes e aritmética de 24 bits para segmentos de 16 Mbytes. O carry e o overflow do bit mais significativo são ignorados. Assim, os endereços circulam dentro de um segmento.

De modo linear, os cálculos de endereços efetivos usam aritmética de 32 bits. O carry e o overflow do bit mais significativo são ignorados. Assim, os endereços circulam com o endereço $2^{32}-1$ seguido do 0.

Os modos de endereçamento da CPU Z80000 são ilustrados na Figura 7.8. Para o registrador de modo de endereçamento, o operando é alocado no registrador de propósito geral específico. Para o modo imediato, o operando é alocado na própria instrução. Para o modo de registrador indireto, o endereço de memória do operando ou endereço de porta de E/S é especificado dentro da instrução. Para o modo de endereçamento indexado, o endereço do operando na memória é calculado somando-se o endereço dado na instrução ao valor de indexação especificado no registrador de propósito geral. Para o modo de endereçamento base, o endereço do operando na memória é calculado somando-se o deslocamento da instrução ao endereço base contido no registrador de propósito geral especificado. Para o modo de endereçamento indexado base, o endereço de memória do operando é calculado somando-se o deslocamento dados na instrução ao endereço base e ao valor de indexação contidos nos registradores de propósitos geral especificados. Para o modo relativo, o endereço do operando é calculado somando-se o deslocamento na instrução ao conteúdo do contador de programa; o operando é alocado no intervalo de endereço de memória de instruções. Para o modo de endereçamento indexado relativo, o operando é alocado no endereço calculado somando-se o deslocamento dado na instrução, o valor de indexação especificado no registrador de propósito geral e o conteúdo do contador de programa; o operando é alocado no intervalo de endereços de memória de instruções.

Modo de endereçamento	Endereçamento do operando			Valor do operando
	Na instrução	Em registrador	Em memória	
R Registrador	NÚMERO DO REGISTRADOR →	OPERANDO		O conteúdo do registrador
IM Imediato	OPERANDO			Na instrução
***IR** Registrador indireto	NÚMERO DO REGISTRADOR →	ENDEREÇO →	OPERANDO	O conteúdo da posição de memória, cujo endereço está no registrador
DA Endereço direto	ENDEREÇO →		OPERANDO	O conteúdo da posição de memória, cujo endereço está contido na instrução
***X** Índice	DO REGISTRADOR ÍNDICE ENDEREÇO BASE →	⊕	OPERANDO	O conteúdo da posição de memória, cujo endereço é o endereço da instrução mais o conteúdo do registrador de índice
***BA** Endereço base	DO REGISTRADOR END. BASE DESLOCAMENTO →	⊕	OPERANDO	O conteúdo da posição de memória, cujo endereço é o conteúdo do registrador de base mais um deslocamento contido na instrução
***BX** Base índice	DO REGISTRADOR END. BASE DESLOCAMENTO ÍNDICE →	⊕	OPERANDO	O conteúdo da posição de memória, cujo endereço é o conteúdo do registrador de base, mais o conteúdo do registrador de índice, mais um deslocamento contido na instrução
RA Endereço relativo	DESLOCAMENTO →	END. DO PC ±	OPERANDO	O conteúdo da posição de memória, cujo endereço é o conteúdo do contador de programa, mais um deslocamento contido na instrução
***RX** Índice relativo	DO REGISTRADOR END. DO PC DESLOCAMENTO ÍNDICE →	⊕	OPERANDO	O conteúdo da posição de memória, cujo endereço é o conteúdo do contador de programa, mais o conteúdo do registrador de índice, mais um deslocamento contido na instrução

* RO e RRO não podem ser utilizados como registradores indiretos, de base, ou índice

Figura 7.8 Modos de endereçamento.

7.5 CONJUNTO DE INSTRUÇÕES

As CPU Z80000 caracterizam-se como um conjunto de instruções rico e poderoso que suporta operações com nove tipos de dados: bit, campo de bit, inteiros sinalizados, inteiros não sinalizados, valores lógicos, endereços, inteiros compactados BCD, pilha e strings. Valores lógicos e inteiros podem ser operandos byte, palavra e palavra longa. Em adição, operações de ponto flutuante são suportadas pelo co-processador. A combinação regular de operações, modos de endereçamento e tipos de dados resultantes em um conjunto de instruções podem ser compilados por uma linguagem de alto nível como C, Ada e Pascal.

O conjunto de instruções pode ser dividido em onze grupos funcionais de instruções:

1. Carga e trocas.
2. Aritmética.
3. Lógica.
4. Controle de programa.
5. Manipulação de bit.
6. Campo de bit.
7. Rotação e deslocamentos (shits).
8. Transferência de blocos e manipulação de string.
9. Saída/entrada.
10. Controle da CPU.
11. Instruções estendidas.

A maioria das instruções tem a forma de byte, palavra ou palavra longa; o sufixo 'B' no mnemônico da instrução indica a forma byte, e o sufixo 'L' mnemônico indica palavra longa.

As instruções de carga e trocas (Tabela 7.1) movem dados entre registradores e memória. As instruções de Load e Load Relative (carga e carga relativa) provêem movimentações básicas, instruções com código especial compactado para a carga de valores constantes pequenos (LDK), carga de valor zero (CLR) ou troca de dois valores (EX). A conversão de instruções permitem valores de um byte, palavra ou palavra longa serem movidos a um destino com tamanho diferente. As instruções de (load multiple – carga múltipla) provêem transferência de blocos entre registradores de propósito geral e memória para salvamento e rearmazenamento eficiente dos valores dos registradores. Mais de dezesseis registradores de palavra longa podem ser carregados em ou de endereços consecutivos na memória com uma instrução simples. Operações com pilha são suportadas pelas

Tabela 7.1 Instruções de carga e troca.

Mnemônico(s)	*Operando(s)*	*Nome da instrução*
CLR,CLRB,CLRL	destino	Limpeza
CVT	destino,fonte	Conversão
CVTU	destino,fonte	Conversão sem sinal
EX,EXB,EXL	destino,fonte	Troca
LD,LDB,LDL	destino,fonte	Carga
LDA	destino,fonte	Carga de endereço
LDAR	destino,fonte	Carga de endereço relativo
LDK,LDKL	destino,fonte	Carga de constante
LDM,LDML	destino,fonte,num	Carga e multiplicação
LDR,LDRB,LDRL	destino,fonte	Carga relativa
POP,POPL	destino,fonte	Desempilha
PUSH,PUSHL	destino,fonte	Empilha

instruções Pop e Push. As instruções de carga de endereços calculam os endereços efetivos do operando e carregam aquele endereço no registrador destino.

O grupo aritmético (Tabela 7.2) suporta adições, subtrações, multiplicações de inteiros sinalizados e não-sinalizados. Aritmética de decimais de código binário é também suportada com instruções de ajuste decimal. São providas instruções que realizam complemento de dois negativos (NEG), comparam dois operandos (CP) e comparam um operando com zero (TESTA). Instruções de incremento e decremento somam ou subtraem uma constante entre 1 e 16 de seus destinos; formas de intertravamento são disponíveis para leitura e recarga de semáforos na memória. As instruções de verificações comparam um operando com os limites inferiores e superiores, gerando um trap quando a fonte está fora dos limites. As instruções de indexação são usadas para computar um indexador dentro de um vetor e comparar o resultado aos limites superiores e inferiores do vetor; novamente um trap é gerado pela condição de fora dos limites.

As instruções de lógica (Tabela 7.3) realizam operações lógicas de todos os bits dentro de um operando. A instrução Test realiza um Or lógico do destino e zero, sinalizando os indicadores apropriadamente.

As instruções de controle de programa (Tabela 7.4) fornecem para controle de programa seqüências de execução de saltos, loops, chamadas de procedimentos e exceções. Instruções de jumps (saltos) condicionais realizam saltos de programa baseados nos estados dos indicadores. Chamadas de procedimentos são suportadas pelas instruções Call, Enter, Exit e Return. A instrução Enter é executada no começo do procedimento para estabelecer o quadro de pilha para aquela chamada de procedimento, incluindo salvamento

de registradores, estabelecimento de ponteiro para quadro e alocação de espaço para variáveis locais; a instrução Exit libera o quadro de pilha no fim do procedimento. Essas instruções fornecem todas as funções iniciais para enlaces de procedimentos em linguagens de alto nível como o C e Pascal. As instruções de decremento e salto se não forem zero são usadas para controlar loops pelo decremento de um contador de loop, testando o contador, e saltos baseados na saída do teste. Os traps de breakpoint (ponto de parada), System Calls (chamadas ao sistema) e instruções de traps condicionais são geradas por condições de trap. Traps de breakpoint são usados para debug. Chamadas do sistema são usadas por rotinas do modo normal para requisitar serviços do sistema operacional, e os traps condicionais permitem ao programador definir condições de trap baseadas no estado dos indicadores.

Tabela 7.2 Instruções aritméticas.

Mnemônico(s)	*Operando(s)*	*Nome da instrução*
ADC,ADCB,ADCL	destino,fonte	Adição com carry
ADD,ADDB,ADDL	destino,fonte	Adição
CHK,CHKB,CHKL	destino,fonte	Verificação
CP,CPB,CPL	destino,fonte	Comparação
DAB	destino	Ajuste decimal
DEC,DECB,DECL	destino,fonte	Decremento
DECI,DECIB	destino,fonte	Decremento interligado
DIV,DIVL	destino,fonte	Divisão
DIVU,DIVUL	destino,fonte	Divisão sem sinal
EXTS,EXTSB,EXTSL	destino	Extensão de sinal
INC,INCB,INCL	destino,fonte	Incremento
INCI,INCIB	destino,fonte	Incremento interligado
INDEX,INDEXL	destino,sub,fonte	Índice
MULT,MULTL	destino,fonte	Multiplicação
MULTU,MULTUL	destino,fonte	Multiplicação sem sinal
NEG,NEGB,NEGL	destino	Negação
SBC,SBCB,SBCL	destino,fonte	Subtração com carry
SUB,SUBB,SUBL	destino,fonte	Subtração
TESTA,TESTAB,TESTAL	destino	Teste aritmético

As instruções de manipulação de bit (Tabela 7.5) permitem ao programador testar, sinalizar ou retirar a sinalização de qualquer bit em qualquer byte, registrador de palavra ou palavra longa ou endereço da memória de dados com uma simples instrução. A instrução de Test Condition Code (Código de Condição de Teste) sinaliza o bit menos significativo de seu destino se os indicadores satisfizerem o código de condição especificado; isto geralmente é usado para avaliar-se condições expressas booleanas. As instruções

de Test e Set são usadas para acessar semáforos que controlam os recursos de compartilhamento em um sistema de multiprocessamento e multitarefa.

As instruções de campo de bit (Tabela 7.6) são usadas para inserir e extrair campos de bits em operandos de palavra longa. Um campo de bit consiste em 1 a 32 bits contínuos. A instrução de Extract Field (extração de bit) extrai um campo de bit do destino e carrega-o em um registrador de palavra longa. A instrução de Insert Field (inserir um campo) insere um campo de bit de um registrador no destino. A posição inicial e tamanho do campo de bit são dados como operandos nessas instruções.

Bytes, palavras e palavras longas dentro de registradores de propósito geral podem ser deslocados ou roteados com as instruções na Tabela 7.7. O conteúdo do registrador pode ser roteado 1 ou 2 bits para a esquerda ou direita com uma instrução simples de rotação. Instruções de rotação de dígito atuam em 4 bits do dígito de um byte do registrador, e é útil para manipulação de dados BCD. Instruções de deslocamento (shift) deslocam o conteúdo do registrador destino para esquerda ou direita em um número de bits especifi-

Tabela 7.3 Instruções lógicas.

Mnemônico(s)	*Operando(s)*	*Nome da instrução*
AND,ANDB,ANDL	destino,fonte	E
COM,COMB,COML	destino	Complemento
OR,ORB,ROL	destino,fonte	OU
TEST,TESTB,TESTL	destino	Teste
XOR,XORB,XORL	destino,fonte	OU exclusivo

Tabela 7.4 Instruções de controle de programa.

Mnemônico(s)	*Operando(s)*	*Nome da instrução*
BRKPT		Breakpoint
CALL	destino	Chamada de procedimento
CALR	destino	Chamada de procedimento relativa
DJNZ,DBJNZ,DLJNZ	r,destino	Decremento e desvio se não zero
ENTER	máscara,tam.	Entrada em procedimento
EXIT		Saída de procedimento
JP	cc,destino	Salto
JR	cc,destino	Salto relativo
RET	cc	Retorno de procedimento
SC	fonte	Chamada de sistema
TRAP	cc,fonte	Trap condicional

cado pelo operando origem. Ambos os deslocamentos lógicos e aritméticos são providos. Deslocamentos aritméticos para a direita preservam o bit de sinal do destino.

As instruções de transferência de blocos e manipulação de string (Tabela 7.8) atuam em strings inteiros de dados em alocação contínua de memória. Strings maiores de 65536 bytes, palavras ou palavras longas podem ser manipuladas por instruções simples. Um bloco de memória pode ser movido para outra área de memória; strings de dados na memória podem ser procurados por um valor ou valores dados; podem ser comparados dois strings, strings de dados podem ser transferidos de um código de 8 bits para outro. O modo de endereçamento de registrador indireto é usado para acessar elementos de um string na memória. Todas as operações podem operar o string em ambas direções; o registro de ponteiro é automaticamente incrementado ou decrementado depois de cada iteração da operação. Um registrador contador, o terceiro operando destas instruções, é decrementado depois de cada iteração da operação. Formas de repetição automaticamente repetem a operação até que o registrador contador seja zero. As formas repetitivas dessas instruções são interrompíveis depois de cada iteração.

As instruções de entrada/saída (Tabela 7.9) transferem dados entre portas de E/S e um registrador ou memória. As instruções de entrada e saída transferem um byte simples, palavra ou palavra longa de dado entre um dispositivo de E/S e um registrador. As instruções restantes são de movimentação de blocos que permitem a transferência de blocos inteiros de dados entre uma porta de periférico e um string de alocação contínua na memória. Como as instruções de movimentação de blocos, strings de memória podem ser acessados em ambas direções, e as formas repetitivas são interrompíveis depois de cada iteração.

As instruções de controle da CPU (Tabela 7.10) são instruções privilegiadas, com exceção da No Operation e instruções de manipulação de indicadores (COMFLG, RESFLG, SETFLG). As instruções de Enable Interrupt e Disable Interrupt (habilitação e desabilitação de interrupção) permitem o controle por software de entradas de interrupções vetorizadas e não-vetorizadas. A instrução de Halt suspende a execução de instruções posteriores na CPU até a próxima interrupção, permitindo que execução da instrução seja sincronizada com um evento externo. A instrução Interrupt Return (interrupção de retorno) é usada para retornar de uma interrupção ou uma rotina de serviço de trap. A instrução Load Control (controle de carga) transfere dados entre um registrador de propósito especial da CPU e um registrador de propósito geral. As instruções de Load Normal Data e Load Normal (carga de dados normais e carga normal) são usadas para acessar os intervalos de endereçamento de memória do modo normal para uma rotina do modo sistema. As instruções de Load Physical Address (carga de endereços físicos) carregam o endereço físico do operando origem para um registrador destino, sinalizando os indicado-

res para indicar proteções de acesso. A instrução Purge cache invalida o conteúdo do cache em pastilha. Várias instruções invalidam as entradas TLB: Purge TLB invalida todas as entradas TLB, e o Purge TLB Normal purga somente as entradas TLB do modo normal, e as instruções Purge TLB Entry podem invalidar somente as entradas associadas com um intervalo de endereçamento de memória particular. As entradas da TLB e purgação de cache são usualmente necessárias quando se muda o mapeamento da memória durante chaveamento de tarefas em sistemas de multitarefas.

Tabela 7.5 Instruções de manipulação de bits.

Mnemônico(s)	*Operando(s)*	*Nome da instrução*
BIT,BITB,BITL	destino,fonte	Testa bit
RES,RESB,RESL	destino,fonte	Limpa bit
SET,SETB,SETL	destino,fonte	Ativa bit
TSET,TSETB,TSETL	destino	Testa e ativa
TCC,TCCB,TCCL	cc,destino	Testa código de condição

Tabela 7.6 Instruções de manipulação de campos de bits.

Mnemônico(s)	*Operando(s)*	*Nome da instrução*
EXTR	destino,fonte,pos.,tam.	Extrai campo
EXTRU	destino,fonte,pos.,tam.	Extrai campo sem sinal
INSRT	destino,fonte,pos.,tam.	Insere campo

Tabela 7.7 Instruções de rotação e deslocamento.

Mnemônico(s)	*Operando(s)*	*Nome da instrução*
RL,RLB,RLL	destino,fonte	Rotação à esquerda
RLC,RLCB,RLCL	destino,fonte	Rotação à esquerda através de carry
RLDB	destino,fonte	Rotação do dígito da esquerda
RR,RRB,RRL	destino,fonte	Rotação à direita
RRC,RRCB,RRCL	destino,fonte	Rotação à direita através de carry
RRDB	destino,fonte	Rotação do dígito da direita
SDA,SDAB,SDAL	destino,fonte	Deslocamento aritmético dinâmico
SDL,SDLB,SDLL	destino,fonte	Deslocamento lógico dinâmico
SLA,SLAB,SLAL	destino,fonte	Deslocamento aritmético à esquerda
SLL,SLLB,SLLL	destino,fonte	Deslocamento lógico à esquerda
SRA,SRAB,SRAL	destino,fonte	Deslocamento aritmético à direita
SRL,SRLB,SRLL	destino,fonte	Deslocamento lógico à direita

Tabela 7.8 Instruções de transferência de bloco e manipulação de strings.

Mnemônico(s)	*Operando(s)*	*Nome da instrução*
CPD,CPDB,CPDL	destino,fonte,r,cc	Compare e decremente
CPDR,CPDRB,CPDRL	destino,fonte,r,cc	Compare, decremente e repita
CPI,CPIB,CPIL	destino,fonte,r,cc	Compare e incremente
CPIR,CPIRB,CPIRL	destino,fonte,r,cc	Compare, incremente e repita
CPSD,CPSDB,CPSDL	destino,fonte,r,cc	Compare string e decremente
CPSDR,CPSDRB,CPSDRL	destino,fonte,r,cc	Compare string, decremente e repita
CPSI,CPSIB,CPSIL	destino,fonte,r,cc	Compare string e incremente
CPSIR,CPSIRB,CPSIRL	destino,fonte,r,cc	Compare string, incremente e repita
LDD,LDDB,LDDL	destino,fonte,r	Carregue e decremente
LDDR,LDDRB,LDDRL	destino,fonte,r	Carregue, decremente e repita
LDI,LDIB,LDIL	destino,fonte,r	Carregue e incremente
LDIR,LDIRB,LDIRL	destino,fonte,r	Carregue, incremente e repita
TRDB	destino,fonte,r	Traduza e decremente
TRDBR	destino,fonte,r	Traduza, decremente e repita
TRIB	destino,fonte,r	Traduza e incremente
TRIRB	destino,fonte,r	Traduza, incremente e repita
TRTDB	fonte1,fonte2,r	Traduza, teste e decremente
TRTDRB	fonte1,fonte2,r	Traduza, teste, decremente e repita
TRTIB	fonte1,fonte2,r	Traduza, teste e incremente
TRTIRB	fonte1,fonte2,r	Traduza, teste, incremente e repita

Tabela 7.9 Instruções de entrada/saída.

Mnemônico(s)	*Operando(s)*	*Nome da instrução*
IN,INB,INL	destino,fonte	Entrada
IND,INDB,INDL	destino,fonte,r	Entrada e decremente
INDR,INDRB,INDRL	destino,fonte,r	Entrada, decremente e repita
INI,INIB,INIL	destino,fonte,r	Entrada e incremente
INIR,INIRB,INIRL	destino,fonte,r	Entrada, incremente e repita
OTDR,OTDRB,OTDRL	destino,fonte,r	Saída, decremente e repita
OTIR,OTIRB,OTIRL	destino,fonte,r	Saída, incremente e repita
OUT,OUTB,OUTL	destino,fonte	Saída
OUTD,OUTDB,OUTDL	destino,fonte,r	Saída e decremente
OUTI,OUTIB,OUTIL	destino,fonte,r	Saída e incremente

Tabela 7.10 Instruções de controle da CPU.

Mnemônico(s)	*Operando(s)*	*Nome da instrução*
COMLFG	indicador	Complemente indicador
DI	int	Desabilita interrupção
EI	int	Habilita interrupção
HALT		Suspende
IRET		Retorno de interrupção
LDCTL,LDCTLB,LDCTLL	destino,fonte	Carga de registrador de controle
LDND,LDNDB,LDNDL	destino,fonte	Carga de dados normal
LDNI,LDNIB,LDNIL	destino,fonte	Carga de instruções normal
LDPND,LDPNI,LDPSD		
LDPSI	destino,fonte	Carga de endereço físico
LDPS		Carga de estado de programa
NOP		Operação nula
PCACHE		Destrua cache
PTLB		Destrua TLB
PTLBEND,PTLBENI,		
PTLBESD,PTLBESI		Destrua entrada de TLB
PTLBEN		Destrua TLB normal
RESFLG	indicador	Limpa indicador
SETFLG	indicador	ativa indicador

O conjunto de instruções básicas pode ser estendido através do uso de co-processadores chamados *unidades de processamento estendido* (EPU). Certos opcodes são reservados para o uso da EPU. As instruções associadas com as EPU são chamadas *instruções estendidas.*

7.6 EXECUÇÃO DE INSTRUÇÕES E EXCEÇÕES

As instruções na CPU Z80000 são altamente canalizadas para maximizar a capacidade de instruções. A Figura 7.9 mostra a canalização síncrona de seis estágios usada para executar as instruções. Os vários estágios canalizados podem ser trabalhados simultaneamente em instruções separadas ou em porções separadas de única instrução complexa. A canalização e o cache em pastilha permitem que instruções simples, tal como carregamentos registrador a registrador, executem em uma taxa de uma instrução por ciclo de processador (dois ciclos de relógio); assim o pico de desempenho da CPU é de 12,5 milhões de instruções por segundo (MIPS) em um relógio de 25 MHz. O desempenho real é tipicamente por volta de 1/30 a 1/50 desse pico para a execução de instruções de multiciclos, contenção dos estágios da canalização para o uso da interface do barramento e acesso à memória principal para o cache e falta de TLB.

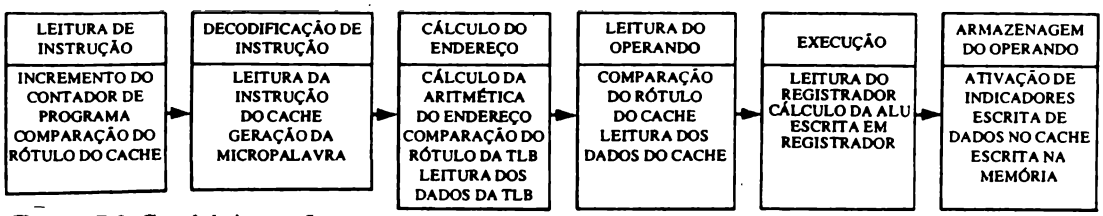


Figura 7.9 Canal de instrução.

Quatro tipos de exceção podem alterar o fluxo normal da execução do programa na CPU Z80000: resets, erros no barramento, interrupções e trap. Os resets ocorrem quando a linha $\overline{\text{RESET}}$ é ativada. Erros no barramento ocorrem quando um hardware externo indica uma condição de erro durante uma transação de barramento. As interrupções são eventos assíncronos detectados em uma das três entradas de interrupção da CPU. Os traps são eventos síncronos detectados internamente durante a execução da instrução.

A CPU responde a um reset lendo valores da posição 2 e 4 da memória física e alocando nos registradores de estado do programa (FCW e PC). Esses valores determinam o endereço inicial e modos de operação para a iniciação da rotina. Certos registradores de controle da CPU são iniciados para estabelecer uma configuração de barramento default, todo o cache e entradas da TLB são invalidados, e o cache e a lógica de gerenciamento de memória são desabilitados.

As respostas para outras exceções envolvem salvamento dos valores de estado do programa corrente e uma palavra identificadora na pilha; algumas exceções salvam informações adicionais de estados. Novos valores de estados do programa são lidos de uma tabela na memória chamada área de estado de programa. Esses novos valores de estado de programa determinam o endereço inicial e os modos de operação da rotina de serviço para aquela exceção. O endereço inicial da área de estado de programa deve ser armazenado no registrador de ponteiro da área de estado de programa da CPU. Cada tipo de exceção tem sua própria e única entrada dentro da área de estado de programa, da qual o estado do programa daquela rotina de serviço pode ser lido. A instrução de retorno de interrupção é usada para terminar a rotina de serviço e o controle de volta para a tarefa de programação que estava executando no momento da exceção.

Uma exceção de erro de barramento provoca imediatamente o término da execução da instrução corrente. O estado dos sinais de estado do barramento e o endereço físico sendo acessado quando o erro ocorre são automaticamente salvos na pilha junto com a informação de estado do programa no momento do processamento da exceção.

São usadas três entradas da CPU Z80000 para detectar três tipos de interrupção: não mascarável, vetorizada e não vetorizada. As interrupções vetorizada e não vetorizada podem ser habilitadas ou desabilitadas por controle de programa, mas a interrupção não

mascarável é sempre habilitada. As interrupções não vetorizadas e não mascaráveis têm uma única rotina de serviço; as interrupções vetorizadas podem ter várias rotinas de serviço potenciais. Quando uma interrupção é detectada, a CPU gera um ciclo de reconhecimento de interrupção para ler uma palavra identificadora do dispositivo de interrupção. Para interrupções vetorizadas, o byte menos significativo do identificador é o vetor usado para selecionar uma das mais de 256 rotinas de serviço possíveis. O identificador é automaticamente armazenado na pilha do sistema com a informação de estado do programa durante o processamento da interrupção.

A CPU Z80000 reconhece dez tipos diferentes de condições de trap: traps de instruções estendidas, traps de instruções privilegiadas, traps de chamada do sistema, traps de tradução de endereços, traps de breakpoint (ponto de parada), traps de instruções não implementadas, traps de PC estranho e traps de trace. Os mecanismos de traps da CPU simplificam debugs de programas e provêem recuperação de software através de detenção de condições de erro em tempo de execução.

O registrador de estado e palavra de controle contêm um bit de habilitação para a arquitetura de processador estendida. Esse bit é usado para informar à CPU da presença de co-processadores no sistema. Se uma instrução estendida é encontrada pela CPU e a arquitetura de processador estendida está desabilitada, ocorre um trap de instrução estendida. Esse trap permite ao software simular a execução da instrução estendida quando os co-processadores não estão presentes.

O trap de instrução privilegiada ocorre quando um programa tenta executar uma instrução privilegiada no modo normal.

O trap de chamada de sistema ocorre quando uma instrução System Call (chamada de sistema) é executada. Esse trap permite programas de modo normal fazerem requisições ao sistema operacional. Um código de requisição de 8 bits que é especificado como um operando imediato na instrução é passado para a rotina de serviço na pilha como parte da palavra identificadora de trap.

Se a lógica de gerenciamento de memória detecta uma entrada de tabela inválida ou uma violação de acesso durante um acesso à memória, o trap de tradução de endereço ocorre. A instrução corrente é terminada imediatamente de uma maneira consistente com as implementações de memória virtual.

O trap de breakpoint ocorre quando uma instrução breakpoint é executada. A instrução breakpoint pode ser usada pelo debugger do software para substituir uma instrução onde o breakpoint é sinalizado.

O trap de aritmética de inteiros ocorre quando um overflow de inteiro, ou verificação de limites ou erro de indexação ocorre durante a execução de instruções de aritmética com inteiros. Os overflows ocorridos durante a execução de uma instrução aritmética causa este trap somente se o trap de overflow de inteiro estiver habilitado na palavra de estado e controle. um erro de verificação de limites é detectado quando uma instrução de Check (verificação) é executada e o operando destino está fora dos limites. Um erro de indexação ocorre quando uma instrução de Index (indexação) é executada e o índice está fora dos limites.

O trap condicional ocorre quando uma instrução de trap é executada e a condição especificada é verdadeira. Esse trap permite que o programador defina condições de trap que detecte erros de tempo de execução definidos pelo usuário.

Se o programa tenta executar um código de instrução que não corresponde a uma das instruções definidas do Z80000, um trap de instruções não implementada ocorre.

O trap de PC ímpar ocorre antes da execução de uma instrução se o contador contém um valor ímpar. Desde que as instruções são sempre alinhadas em limites de palavra na memória, um valor ímpar no contador de programa indica que um erro ocorreu.

O trap de trace ocorre antes da execção de cada instrução se os traps de trace estão habilitados na palavra de estado e controle. O trap de trace provê um meio de debug passo a passo do programa.

Durante o processamento de exceções, os valores de estados do programa, a palavra de identificação e outras informações de estados são salvos na pilha do sistema. Se a lógica de gerenciamento de memória detecta uma violação de acesso durante esse processo de salvamento dos estados, o ponteiro da pilha do sistema é restaurado com o valor anterior à ocorrência de exceção, e o overflow de pilha é usado para salvar a informação de estados para a exceção. O endereço físico da pilha é determinado pelo conteúdo do registrador de ponteiro de overflow de pilha da CPU. São lidos então novos estados de programa em posições da área de estados de programa dedicada a esse tipo de erro de pilha do sistema.

7.7 MULTIPROCESSAMENTO

A arquitetura da CPU Z80000 provê suporte para quatro diferentes tipos de configurações de multiprocessador: processadores escravos, CPU ligadas amarradas, CPU múltiplas acopladas e co-processadores (Figura 7.10).

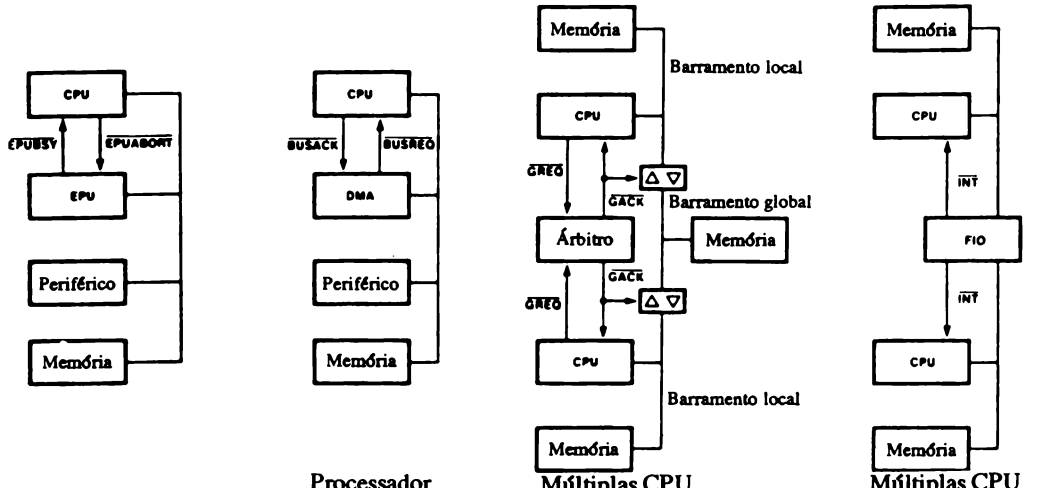


Figura 7.10 Configurações de multiprocessadores.

Processadores escravos, como as controladoras DMA, realizam funções dedicadas assíncronas para a CPU. A CPU e processadores escravos compartilham um barramento local. A CPU é o mestre default; os processadores escravos devem requisitar o uso do barramento para a CPU. A CPU Z80000 tem uma entrada de requisição de barramento que os processadores escravos podem usar para requisitar o barramento, e uma saída de reconhecimento de barramento usada para informar os processadores escravos quando a CPU cede o barramento em resposta à requisição.

Sistemas de multiprocessamento parelhados amarrados são sistemas onde CPU múltiplas executam fluxos de instruções independentemente com suas próprias memórias (local) e se comunicam através da mesma memória compartilhada (global). Cada CPU é o mestre default de seus barramentos locais, mas o barramento global mestre é escolhido por arbitrador externo. Bits de controle no registrador de controle de interface da CPU permitem que o programador habilite a CPU para esse modo de multiprocessamento e escolha o intervalo de endereços dedicados ao uso local e global. Uma saída de requisição global ($\overline{\text{GREQ}}$) e uma entrada de reconhecimento ($\overline{\text{GACK}}$) são oferecidas para implementar essa configuração de multiprocessamento. Enquanto controlando seu barramento local, a CPU pode atender iniciações de transações no barramento global. Para cada transação na memória ou E/S, a CPU verifica se o barramento global está requisitado. Antes de efetuar uma transação no barramento global, a CPU deve insistir em requisitá-lo através de $\overline{\text{GREQ}}$ e esperar por uma resposta ativa $\overline{\text{GACK}}$. Quando um $\overline{\text{GACK}}$ é respondido, a CPU pode efetuar a transação.

CPU parelhadas comunicam através de memória de múltiplos portos ou dispositivos periféricos, tal como a unidade de entrada/saída FIFO Z8038. A CPU provê informações de estado externo sobre referências de memória interconectada que podem ser usadas para controlar memória de múltiplas referências.

A arquitetura da CPU Z80000 suporta o uso de co-processadores chamados unidades de processamento estendido (EPU). As EPU realizam funções especiais em paralelo com a CPU. As EPU conectam diretamente o Z-Bus e efetuam operações com dados residentes nos registradores internos. Mais que quatro EPU podem ser conectadas a uma única CPU. O processador aritmético de ponto flutuante Z8070 é um exemplo de EPU.

Quatro tipos de instruções estendidas são usados para controlar as EPU:

1. transferência de dados entre uma EPU e memória;
2. transferência de dados entre uma EPU e a CPU;
3. transferência de estados dos indicadores entre uma EPU e a CPU;
4. operações internas da EPU.

Cada EPU monitora continuamente o fluxo de instruções, capturando e executando instruções determinadas para isso. A CPU e a EPU cooperam na execução de instruções estendidas. Quando uma CPU recebe uma instrução estendida (e a arquitetura de processador estendido está habilitada), a CPU transmite a instrução para todos os co-processadores (broadcast) no sistema. Se a instrução envolve uma transferência de dados (dos três primeiros tipos de instrução estendida dos quatro citados acima), a CPU é responsável por gerar as transações de barramento apropriadas, incluindo cálculos de endereços efetivos. As EPU monitoram a atividade do barramento, capturando ou fornecendo dados nos momentos apropriados, como se elas fossem parte da CPU. Mais de dezesseis palavras podem ser transferidas para ou de uma EPU através de uma única instrução.

Enquanto a EPU executa uma instrução estendida de operação interna, a CPU pode ler e executar instruções subseqüentes. Assim, as EPU e a CPU podem operar em paralelo. O sinal $\overline{\text{EPUBUSY}}$ (barrado) é usado para sincronizar atividades da CPU e EPU no caso de sobreposição de instruções estendidas. Se uma outra instrução estendida para uma EPU é encontrada enquanto aquela EPU ainda está ocupada processando uma instrução estendida anterior, a EPU insiste na ativação do sinal $\overline{\text{EPUBUSY}}$ até que ela esteja pronta para aceitar uma instrução nova. Enquanto a entrada $\overline{\text{EPUBUSY}}$ está ativa, a CPU não pode iniciar nenhuma nova transação no barramento. Se a CPU detectar um trap de tradução de endereço durante a execução de uma instrução estendida, a saída $\overline{\text{EPUABORT}}$ insistirá para informar à EPU selecionada para abortar a execução da instrução.

Se as EPU não estiverem presentes no sistema, a atividade da EPU pode ser simulada por software usando-se o trap de instrução estendida.

7.8 MEMÓRIA CACHE

A CPU Z80000 contém 256 bytes de memória de alta velocidade que podem ser configurados como um cache associativo ou posições de memória física fixas dedicadas. Quando são configurados como cache, essa memória pode ser um cache para leitura de instruções, referência a dados, ou ambos, sendo determinada pelo conteúdo pelo registrador de palavra longa de controle e configuração do sistema da CPU. Páginas individuais podem ser configuradas como cache habilitada ou cache não habilitada na lógica de gerenciamento de memória. Quando configurado como caches, o conjunto de posições de memória mapeado no cache em um dado momento é determinado pela ação do programa sendo executado, e as posições de memória que foram acessadas mais recentemente são guardadas no cache. A memória lida nas posições já armazenadas no cache não geram transações de barramento externo e, portanto, são mais rápidas que o acesso à memória externa. Assim, o uso de cache resulta em execução mais rápida do programa.

A organização do cache é ilustrada na Figura 7.11. Memória cache é disposta em dezesseis filas de oito palavras cada, com um rótulo de endereço de 28 bits, oito bits de validade associados com cada fila. Cada fila do cache contêm espaço para oito palavras contínuas, onde o rótulo de endereço indica o endereço físico armazenado naquela fila. Para cada acesso à memória, os rótulos são verificados para ver se aquela posição foi correntemente mapeada no cache. Um bit de validade associado com cada palavra na fila indica se aquela palavra contém uma cópia dos dados na posição física de memória associada. O cache está totalmente associado quando nenhuma posição de memória pode ser associada a qualquer fila.

As linhas do cache podem ser realocadas usando-se o último algoritmo recentemente usado (LRU). Se um cache habilitado de leitura de transações acessa uma memória física não mapeada correntemente no cache, a fila no cache que foi a última mais recentemente acessada é selecionada para guardar o novo dado lido.

Quando o cache não está habilitado, cada acesso à memória é uma "colisão de cache" ou uma "falta de cache". Para operações de leitura, uma colisão de cache significa que dados válidos para aquela referência de memória estão no cache e não foi requisitada nenhuma transação externa. Uma falta de cache na leitura significa que acesso a dados não é válido no cache, e no caso será gerada uma transação de barramento externa, e os conteúdos do cache serão atualizados para a inclusão de dados novos. Isto pode envolver re-

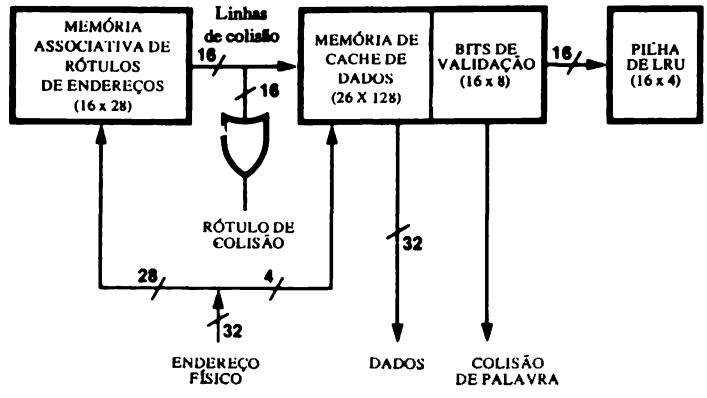


Figura 7.11 Organização do cache.

posição de uma fila nova no cache usando-se algoritmo LRU. Para operações de escrita, uma colisão de cache significa que a posição acessada está em uma fila do cache. É gerada uma transação de barramento externa, e o dado é escrito em ambos, memória externa e cache. Uma falta de cache na escrita resulta em uma transação de barramento externa e não tem efeito no cache. Os dados são sempre escritos na memória externa independentemente do estado do cache, garantindo que a memória externa sempre retenha os dados válidos.

Se as memórias do modo-burst são empregadas no sistema, a CPU usará transações do modo-burst para a pré-leitura de um bloco de memória no cache para a falta de cache em uma instrução de leitura. As transações de modo-burst são sempre usadas para armazenar e ler operandos de dados quando é necessária mais que uma transferência, como com operandos não alinhados, instruções de string e instruções de *multiple load* (múltiplo carregamento).

Um bit de controle no registrador de palavra longa de controle e configuração pode ser usado para desabilitar o algoritmo de realocação do cache, colocando a memória em pastilha em modo de endereçamento fixo. Assim, posições selecionadas podem ser fechadas para acessos em pastilha rápidos.

7.9 INTERFACE EXTERNA

Para completar um sistema de computador, a CPU Z80000 precisa interfacear com outros dispositivos, incluindo memórias, periféricos, controladoras DMA, co-processadores e outras CPU. A Zilog estabeleceu o Z-Bus como convenção para descrever os

sinais envolvidos nos componentes interfaceados de um sistema de microprocessadores. O Z80000 é compatível com a família de processadores Z-Bus, pastilhas de suporte e dispositivos periféricos (Tabela 7.11). Os periféricos de multifunções Z-Bus são extensivamente programáveis, de forma que cada um possa ser precisamente adequado para cada aplicação.

A interface externa da CPU Z80000 tem 59 linhas de sinal e 4 conexões para fornecimento de força (Figura 7.12). Um resumo das funções dos pinos é dado abaixo:

AD0-AD31, Address/Data (bidirecional, ativo alto, 3 estados).

Multiplexa tempo para linhas de barramento de endereços e dados que conduzem endereços ou dados de memória ou portas de E/S durante transações de barramento.

$\overline{AS}$, Address Strobe (saída, ativo baixo, 3 estados). Um sinal de timing de barramento que elevando a borda marca o começo de uma transação de barramento e indica que o endereço e os sinais de estado do barramento são válidos.

$\overline{BRST}$, Burst (saída, ativo baixo, 3 estados). Sinal de estado que indica que o dispositivo de memória pode suportar transferências em modo-burst.

Tabela 7.11 Componentes atuais da família do barramento Z

CPU do barramento Z:	
Z80000	CPU de 32 bits
Z8001/2/3/4	CPU de 16 bits
Z8	Família de microprocessadores de 8 bits em única pastilha
Dispositivos de suporte a processador do barramento Z:	
Z8070	Unidade aritmética em ponto flutuante
Z8016	Controlador de DMA de dois canais
Z8581	Gerador e controlador de relógio
Periféricos do barramento Z:	
Z8030	Controlador de comunicação serial de multiprotocolos, de dois canais
Z8031	Controlador de comunicação assíncrona, de dois canais
Z8036	Unidade de E/S paralela e contador/temporizador
Z8038	Unidade de interface de E/S FIFO
Z8060	Unidade de buffer FIFO
Z8065	Processador de erro de burst
Z8068	Processador de cifragem de dados
Z8052	Controlador de vídeo
Z8090,Z8094	Controlador universal de periféricos

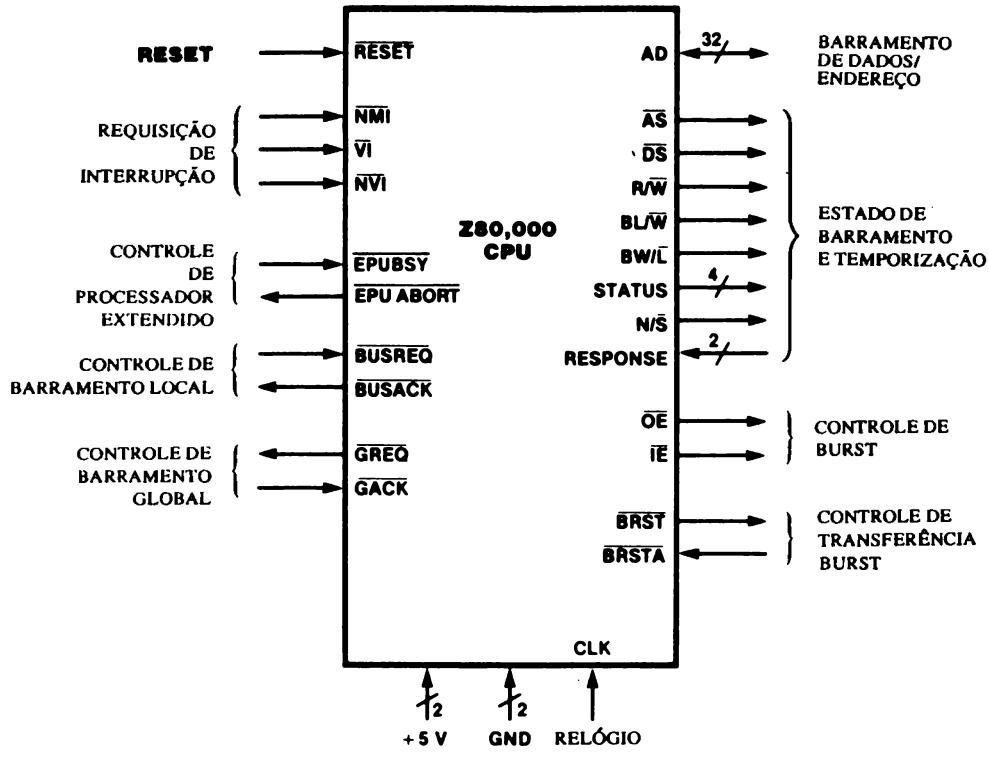


Figura 7.12 Pinagem.

$\overline{\text{BUSREQ}}$, Bus Request (requisição de barramento – entrada, ativo baixo). Sinal que indica que uma requisição de barramento obteve ou está tentando obter controle do barramento local.

$\overline{\text{BUSACK}}$, Bus Acknowledge (reconhecimento de barramento – saída, ativo baixo). Sinal que indica que a CPU liberou o controle do barramento local em resposta a uma requisição de barramento.

$\overline{\text{BL/W}}$, BW/$\overline{\text{L}}$, (saída, 3 estados). Sinal de estado que indica o tamanho dos dados da transação corrente (byte, palavra, palavra longa).

CLK, Clock (entrada). Entrada de relógio usado para gerar todo o timing da CPU.

$\overline{\text{DS}}$, Data Strobe (saída, ativo baixo, 3 estados). Sinal que indica o time de todas as transferências de dados.

$\overline{\text{EPUBUSY}}$, EPU Busy (EPU ocupada – entrada, ativo baixo). Sinal que indica quando

uma EPU está ocupada, usado para sincronizar as atividades da CPU e EPU.

$\overline{\text{EPUABORT}}$, EPU Abort (aborto para EPU – saída, ativo baixo). Sinal que indica que a CPU está abortando uma instrução estendida devido a um trap de tradução de endereço.

$\overline{\text{GACK}}$, Global Acknowledge (reconhecimento global – entrada, ativo baixo). Sinal que indica que a CPU liberou o controle do barramento global.

$\overline{\text{GREQ}}$, Global Request (requisição global – saída, ativo baixo, 3 estados). Sinal que indica que a CPU obteve ou está tentando obter o controle do barramento global.

$\overline{\text{IE}}$, Input Enable (habilitador de entrada – saída, 3 estados, ativo baixo). Sinal que indica que a direção de transferência no barramento de dados/endereços é em direção a CPU. Esse sinal é usado tipicamente para controlar buffers de barramento.

$\overline{\text{NMI}}$, Non-maskable Interrupt (interrupção não mascarável – entrada, edge-activated). Uma transição de 1 para 0 nessa linha requisita uma interrupção não mascarável.

$\overline{\text{NVI}}$, Non-vectored Interruption (interrupção não vetorizada – entrada, ativo baixo). Um sinal 0 nessa linha requisita uma interrupção não vetorizada.

N/$\overline{\text{S}}$, Normal/System Mode (modo normal/sistema – saída, 0 = modo sistema, 3 estados). Sinal de estado que indica o modo de operação corrente da CPU.

$\overline{\text{OE}}$, Output Enable (habilitador de saída – saída, ativo baixo, 3 estados). Sinal que indica que a direção de transferência no barramento de dados/endereços é para fora da CPU. Esse sinal é usado tipicamente para controlar buffers de barramento.

R/$\overline{\text{W}}$, Read/Write (leitura/escrita – saída, 0 = write, 3 estados). Sinal de estado que indica a direção da transferência de dado durante uma transação de barramento.

$\overline{\text{RESET}}$, Reset (reiniciar – entrada, ativo baixo). Reinicia a CPU.

Tabela 7.12 Codificação das entradas de resposta

RSP0	RSP1	*Respostas*
1	1	Pronto
0	1	Erro de barramento
1	0	Tentativa de barramento
0	0	Espera

RSP0-RSP1, Response (respostas – entradas). Essas linhas codificam respostas da memó-

ria externa ou periféricos para transações iniciadas pela CPU, como mostrado na Tabela 7.12.

Tabela 7.13 Codificação das linhas de estado ST3-ST0

ST3-ST0	*Definição*
0000	Operação interna
0001	Transferência de dados CPU-EPU
0010	Transação de E/S
0011	Suspenso
0100	Transferência de instruções CPU-EPU
0101	Reconhecimento de interrupção não mascarável
0110	Reconhecimento de interrupção não vetorada
0111	Reconhecimento de interrupção vetorada
1000	Transferência de dados CPU-memória mantenível em cache
1001	Transferência de dados CPU-memória não mantenível em cache
1010	Transferência de dados EPU-memória mantenível em cache
1011	Transferência de dados EPU-memória não mantenível em cache
1100	Leitura de instrução mantenível em cache
1101	Leitura de instrução não mantenível em cache
1110	Reservado
1111	Transferência de dados CPU-memória interligada

ST0-ST3, Status (estados – saída, ativo alto, 3 estados). Essas linhas codificam o tipo de transação que está ocorrendo no barramento, como mostrado na Tabela 7.13.

$\overline{VI}$, Vectored Interrupt (interrupção vetorizada – entrada, ativo baixo). Um 0 nessa linha requisita uma interrupção vetorizada.

A CPU Z80000 efetua transações em sua interface externa durante a leitura de instruções, leitura e escrita de operando de dados, processamento de exceções e operações de gerenciamento de memória. A CPU também efetua operações internas e transações de parada (halt), que não envolvem transferência de dados.

O registrador de controle de hardware da CPU (HICR) especifica as características da interface de barramento, incluindo velocidade do barramento, tamanho dos dados da memória que passam e o número de estados de espera automáticos.

Com intenção de especificar a configuração do barramento, a memória física é dividida em duas seções, M0 e M1, selecionadas pelo bit 30 do endereço físico. Similarmente o intervalo de E/S físico é dividido em duas seções, E/S0 e E/S1. Por exemplo, um sistema poderia ter uma ampla EPROM vagarosa de 16 bits em M0, e uma ampla RAM de

32 bits em M1. O conteúdo do HICR especifica os seguintes parâmetros de interface de barramento:

M_0 contador de espera – o número de estados de espera automáticos (0 a 7) inseridos durante os acessos a M_0.

M_0 tamanho de dados – tamanho dos dados que passam (16 ou 32 bits) para acessar M_0.

M_1 contador de espera – o número de estados de espera automáticos (0 a 7) inseridos durante o acesso a M_1.

M_1 tamanho dos dados que passam – o tamanho dos dados que passam (16 ou 32 bits) para acessar M_1.

E/S_0 contador de espera – o número de estados de espera automáticos (0 a 7) inseridos durante o acesso ao E/S_0.

E/S_1 contador de espera – o número de estados de espera automáticos (0 a 7) inseridos durante o acesso ao E/S_1.

Contador de espera de reconhecimento de interrupção – o número de estados de espera (0 a 7) inseridos durante uma transação de reconhecimento de interrupção.

Velocidade de relógio do barramento – a freqüência do relógio do barramento (1/2 ou 1/4 da freqüência de relógio da CPU).

Modo sobreposto de EPU – grau de sobreposição de operações da CPU e EPU.

Reconhecimento de endereço mínimo – quando habilitado certifica que um reconhecimento de endereço é gerado no mínimo uma vez a cada 16 ciclos de relógio de barramento.

Habilitação global e endereço local – habilita os modos de barramento local/global e determina os intervalos de endereços no barramento local e global.

As transações no barramento iniciam quando um reconhecimento de endereço ($\overline{AS}$) é enviado e então negado. Se um endereço é requisitado, o endereço é válido no zero $\overline{AS}$. As linhas de estado do barramento também são válidas na borda positiva do $\overline{AS}$. ST0 até ST3 indicam o tipo de transação que está ocorrendo. $R/\overline{W}$ indica a direção de transferência, um $BW/\overline{L}$ e um $BL/\overline{W}$ indicam o tamanho dos dados sendo transferidos, um $N/\overline{S}$ indica o modo de operação da CPU corrente. O sinal de strobe de dado $N/\overline{S}$ é usado para temporizar a transferência de dados. Para escritas, um ($\overline{DS}$) ativo indica.que a CPU alocou dados válidos no barramento de dados/endereços. Para leituras, a CPU dispõe 3 estados

do barramento de endereços/dados antes de ativar o $\overline{DS}$, assim o dispositivo endereçado pode conduzir os dados no barramento.

Durante cada transação de dados, o dispositivo correspondente retorna um código para CPU na entrada de resposta indicando que está pronto, esperando, erro de barramento, ou reiniciação de barramento. As linhas de resposta são testadas pela CPU prioritariamente para testar dados durante um ciclo de leitura ou negar um $\overline{DIS}$ durante um ciclo de escrita. A resposta de pronto indica o sucesso na execução da transação. A resposta de espera indica que o dispositivo que responderá precisa de mais tempo para completar a transação; outro ciclo de barramento, chamado estado de espera, é então somado à transação antes de testar a resposta novamente. Não há limites para o número de estados de espera que possam ser somados a uma única transação via programação do HICR. Estados de espera programados são inseridos na transação antes de testar as linhas de resposta. As respostas de erros de barramento ativam a exceção de erro do barramento, que é processado de uma maneira similar a uma interrupção ou trap. A resposta de reiniciação de barramento faz a CPU tentar efetuar aquela transação novamente.

Transações de leitura de memória, por default, transpõe dois períodos de relógio de barramento. O endereço e estado são emitidos durante o primeiro período de relógio, e as linhas de resposta e os dados são testados durante o segundo período de relógio. Leituras de dois ciclos são propósitos para uso com memórias rápidas, tal como um cache externo. Velocidades de memória mais moderadas requerem o uso de estados de espera durante a operação de leitura. Leituras em E/S e escritas na memória, por default, usam três ciclos de relógio de barramento.

Para aumentar a banda de transação de memória, podem ser usadas transações de modo-burst. Transações de modo-burst usam sinais múltiplos de strobe de dados depois de um strobe de dados simples para transferir dados em endereços consecutivos de memória. A CPU usa transações de modo-burst para pré-leitura de um bloco para a falta de cache na leitura de uma instrução, ou para ler ou escrever dados operandos para instruções com múltiplas transferências. No início de uma transação do modo-burst, a CPU ativa a saída $\overline{BRST}$. O sinal de reconhecimento $\overline{BRSTA}$ é testado em linhas de respostas para indicar se a memória suportará uma transação de modo-burst antes que a CPU prossiga com o acesso em modo-burst.

7.10 DESENVOLVIMENTO DE SISTEMAS E LINGUAGENS

Desde que a CPU Z80000 é compatível com o microprocessador Z8000 de 16 - bits, as ferramentas de desenvolvimento já disponíveis para a família Z8000 podem ser

aplicadas facilmente para os projetos baseados no Z80000. Novas ferramentas otimizadas para o Z80000 são também disponíveis.

A família 8000 de sistemas da Zilog de microcomputadores de alto desempenho provê um meio de desenvolvimento de software ideal. O hardware do sistema 8000 combina a CPU Z8000 com disco Winchester de alto desempenho e discos inteligentes e controladoras de fita. Os membros da família estão entre sistemas de 8 usuários com 512 kbytes de memória principal e um disco Winchester de 23 Mbytes para sistemas de 40 usuários com 4 Mbytes de memória principal e um disco Winchester de 168 Mbytes. O sistema operacional UNIX da Zilog foi projetado especificamente para desenvolvimento de software e teste. Numerosas ferramentas de desenvolvimento incluindo depuradores, bibliotecas, compiladores e montadores permitem o desenvolvimento de código para um número diferente de tipos de processadores. Os enlaces seriais padrão RS-232 no sistema de microprocessadores 8000 provêem envio e recepção de programas de hospedeiros de programas para emulação de sistemas de hardware. O sistema 8000 caracteriza um passo para um fácil alcance de todo um sistema de 32 bits baseado no Z80000. Além disso, a Zilog está fornecendo suporte para equipamentos originais fabricados que desejem portar o sistema Unix V para a CPU Z80000.

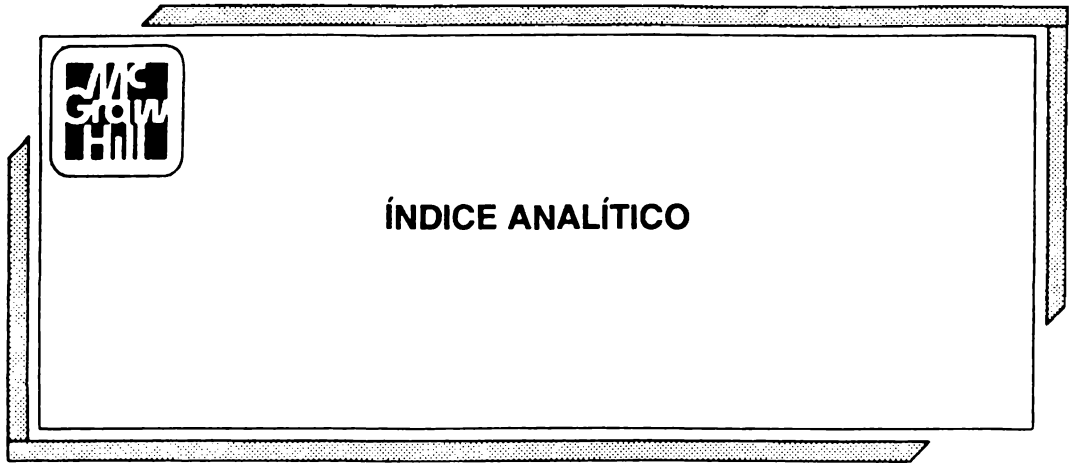
As linguagens correntes disponíveis para o Z8000 (e, portanto, para o Z80000) incluem Ada, C, Pascal, FORTRAN, COBOL e BASIC. As versões desses compiladores otimizadas para o Z80000 estão prestes a estar ou já estão disponíveis. O cross-assembler Z80000 suporta macro ou condicionais assembly, e é disponível para os sistemas 8000, mestres compatíveis VAX ou IBM-PC.

Ao lado de sistemas de desenvolvimento, emuladores e software de desenvolvimento disponíveis da Zilog, numerosos produtos de suporte são disponíveis através de representantes.

7.11 SUMÁRIO

Em resumo, a CPU Z80000 encontra e atravessa as requisições de sistemas de microprocessadores de fins médios e altos. Suas características de arquitetura abundante permitem seu uso em aplicações previamente legadas somente a minicomputadores e computadores de grande porte, tal como automação de escritório, estações de trabalho de engenharia, processamento de gráfico, processamento de sinais digitais, robótica, sistemas inteligentes e reconhecimento de línguas e sínteses. A arquitetura do Z80000 torna disponível um largo campo de operações de configuração para o projetista, permitindo integração de aplicações desde controladoras dedicadas até grandes sistemas de multiprocessa-

mento, interface de co-processadores, alta velocidade de relógio e um grande conjunto de instruções, tudo contribui para alcançar-se taxas de capacidade previamente inantigíveis. Desenvolvimento de programas de software é facilmente feito com a sofisticada arquitetura da CPU. A família de periféricos Z8000 bem estabelecida, processadores e dispositivos de suporte e os sistemas de desenvolvimento da família sistema 8000 e desenvolvimento de software são totalmente compatíveis com o microprocessador de 32 bits Z80000, o último da linha Zilog de componentes VLSI estado da arte de alto desempenho.



ÍNDICE ANALÍTICO

Composição e Arte-Final:
JAG Composição Editorial e Artes Gráficas Ltda.
Praça F. Roosevelt, 208 - 8º andar
Tel. (011) 255-5694 - São Paulo

Impresso na Prol editora gráfica ltda.
03043 Rua Martim Burchard, 246
Brás - São Paulo - SP
Fone: (011) 270-4388 (PABX)
com filmes fornecidos pelo Editor.

OUTROS LIVROS NA ÁREA

Andrews	– 65C02/6502B – Assembler
Carvalho	– Assembler para o TK 90X, TK 95
Carvalho	– Assembler para o MSX
Cereda/Maldonado	– Introdução ao Fortran 77 para Microprocessadores
Ciarcia	– Construa seu próprio Microcomputador Z80
Cramer	– 68000 Microprocessador
Malvino	– Microcomputadores e Microprocessadores
Malvino	– Eletrônica Digital
Morgan	– 8086/8088 – Manual do Microprocessador de 16 Bits
Mottola	– Linguagem de Programação Assembly para Apple II – 6502
Osborne	– Introdução aos Microcomputadores – vols. 0 e 1
Shimizu	– Programação Assembler – Z80, 6502, 8080, 8085
Shimizu	– Programação Assembler para Microprocessadores 68000, 68010, 68020
Taub	– Circuitos Digitais e Microprocessadores
Taub	– Eletrônica Digital
Tokheim	– Introdução aos Microprocessadores
Tokheim	– Princípios Digitais
Tokheim	– Circuitos Eletrônicos e de Microcomputadores

0-07-450282-4