UMA PROPOSTA DE PROCESSAMENTO PARALELO ATRAVÉS DE UMA ARQUITETURA A MICROPROCESSADORES

José Motta da Rocha Lopes

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS (M.Sc.)

Aprovado por:

Sueli Mendes dos Santos
(Presidente)

Leslie Afrânio Terry

Edil Severiano Tavares Fernandes
Edil Severiano Tavares Fernandes

Michael Anthony Stanton

RIO DE JANEIRO, RJ - BRASIL
NOVEMBRO DE 1982

MOTTA, JOSÉ

Uma Proposta de Processamento Paralelo Através de uma Arquitetura a Microprocessadores [Rio de Janeiro] 1982.

VIII, 145p., 29,7cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1982).

Tese - Univ. Fed. Rio de Janeiro, CEPEL

1. Sistema Distribuído Baseado em Microprocessadores. I. COPPE/UFRJ II. Título (Série)

A meus pais
e Marília

AGRADECIMENTOS

A Leslie Afranio Terry pela orientação e por todo o esforço dispendido no desenvolvimento deste trabalho.

A Sueli Mendes dos Santos pela orientação e pelos conhecimentos transmitidos durante os cursos.

A Valmir Carneiro Barbosa pela participação nas discussões que definiram o projeto.

A Cesar Augusto Boavista de Freitas pela revisão do trabalho.

A Rosangela Silva Torres e a Leena Motta da Rocha Lopes pelos trabalhos de datilografia e confecção gráfica da tese.

Ao CEPEL por ter possibilitado o desenvolvimento do presente trabalho através de incentivos e recursos.

SINOPSE

Este trabalho foi motivado pela crescente importância que os projetos de software básico para sistemas distribuídos com microprocessadores tem representado para o desenvolvimento de sistemas de computação mais eficientes.

É proposta a adaptação de um sistema de desenvolvimento de programas, que opera em microcomputadores de pequeno porte, de forma a permitir o processamento científico necessário a um Centro de Controle e Supervisão para Sistemas Elétricos de Potência. Os principais aspectos considerados na máquina estendida são:

- A manutenção da portabilidade existente no sistema original;

- A interligaçao de microprocessadores por intermédio de uma topologia fortemente conectada;

- A substituição do "stack" linear por uma estrutura mais adequada ao processamento paralelo;

- A introdução de técnicas mais otimizadas de gerenciamento de memória.

Um protótipo para emulação de parte das funções especificadas do processador virtual foi construído a partir do microprocessador 8086. Na implementação das etapas futuras do projeto é prevista a utilização de componentes mais sofisticados.

ABSTRACT

This work was motivated by the growing importance of distributed systems software design in the development of more efficient computer systems.

An adaptation of a development system for sequential programs which was designed for small microcomputer environments is proposed. The new architecture will support the scientific computation that is peformed in Supervisory and Control Systems for Electric Power Networks. The main aspects considered in the extended machine are:

- The maintenance of the portability of the original system;

- The use of microprocessors connected in a tightly coupled topology;

- The replacement of the linear stack by a new structure, more adequate to parallel processing;

- The design of improved techniques for memory management.

A prototype based on a 8086 microprocessor emulates some of the functions that were specified for the virtual processor. In the implementation of the future stages of the project the use of more sofisticated components is antecipated.

ÍNDICE

I. INTRODUÇÃO

As ciências associadas à computação tem, inegavelmente, evoluído muito rapidamente desde a sua criação. Contudo, observando-se um passado recente, verifica-se que os ganhos em eficiência que foram introduzidos nas máquinas, vem sendo consumidos por tarefas que tem igualmente crescido em complexidade. Esta constatação é confirmada, por exemplo, pela verificação de que o tempo de resposta dos sistemas operacionais multi-usuários não tem variado tanto quanto o potencial dos computadores que os executam.

Os microprocessadores preencheram, durante a última década, espaços que até então estavam inexplorados ou ocupados de forma ineficiente por minicomputadores ou circuitos com lógica discreta. Atualmente, após a consolidação deste mercado, buscam-se novas técnicas que possam alargar o espectro de utilização destes componentes. Uma alternativa bastante promissora tem sido a exploração de estruturas compostas por vários elementos inteligentes, não muito complexos, que, através de cooperação mútua, tenham capacidade para resolver problemas de grande porte. Os Capítulos II e III apresentam a descrição de diversas arquiteturas adequadas ao processamento paralelo.

O presente trabalho faz parte de um programa estabelecido pelo CEPEL para o desenvolvimento de Sistemas de Supervisão e Controle para Sistemas Elétricos de Potência. Os resultados produzidos por este programa tem atendido, até o presente momento, às diretrizes básicas estabelecidas, que são:

- Aquisição de grandezas representativas do estado do sistema elétrico, junto aos processos que estão sendo supervisionados;

- Transmissão das informações aquisitadas através de uma estrutura hierarquizada de centros de tratamento de informação e

- Interação com o elemento humano, visando a exteriorização do

estado do sistema e a aceitação de comandos que o modifiquem.

Nesta linha, foram projetados e construídos um Terminal Remoto para Aquisição de Dados (70, 71) e um Centro de Supervisão Regional (72). Ambos os sistemas estão sendo atualmente industrializados, acarretando a economia de um volume significativo de divisas para o setor, através da nacionalização de tecnologia e equipamentos. Em termos brasileiros, estes projetos ganham ainda maior importância, quando se constata a gama de possíveis aplicações que podem utilizá-los. Como exemplo, podem ser citados o controle e supervisão de siderúrgicas, refinarias, centrais de abastecimento de água e processos industriais de uma forma geral. Todas estas atividades oneram substancialmente a balança de importações atual.

O desenvolvimento deste trabalho foi motivado pela necessidade de promover a expansão vertical da linha anteriormente citada, pois os modernos centros de controle exigem cada vez mais a incorporação de funções complexas de processamento. Estas funções incluem estimadores de estado, configuradores, fluxos de potência, etc.. Estas tarefas vem sendo realizadas por minicomputadores de "grande porte" (como o VAX 11/780), pois demandam uma capacidade de processamento superior àquela permitida pelos microcomputadores comerciais atuais (1).

Como ponto de partida para a realização do software do projeto foi utilizado um sistema construído pela Universidade da Califórnia em San Diego (UCSD), capacitado para fornecer aos usuários de microcomputadores facilidades de edição, compilação e execução de programas sequenciais. O Sistema-p, como é chamado, é na sua maior parte escrito em linguagem de alto nível e pode ser facilmente adaptado a diversos ambientes de computação. Um pseudo-processador, que executa um código intermediário (código-p), garante que somente uma pequena parte do sistema (correspondente à emulação da máquina virtual) esteja diretamente relacionado com o hardware específico de uma determinada implementação.

A adaptação do Sistema-p para atender aos requisitos do projeto

foi dividida em duas etapas: adaptação da linguagem Pascal UCSD à programação concorrente e a especificação de uma arquitetura que emulasse o pseudo-processador do Sistema-p. A primeira etapa, que encontra-se descrita em Barbosa et al (69), adicionou diversos construtos à linguagem para possibilitar o sincronismo entre processos concorrentes e garantir a exclusão mútua aos recursos comuns por eles compartilhados. A etapa aqui descrita inclui o projeto de uma arquitetura baseada em microprocessadores dispostos segundo uma topologia fortemente conectada, além da especificação de um interpretador capaz de executar o pseudo-código da máquina-p estendida. As principais alterações realizadas na versão original do sistema, descritas nos dois últimos capítulos deste trabalho, são:

- a extensão do stack linear da máquina-p para uma estrutura em árvore que permitisse o acesso simultâneo de diversos processos a regiões comuns de memória;

- a partição da memória principal em segmentos distintos, visando a separação das áreas de código e dados estáticos e dinâmicos;

- a introdução de memória virtual, para armazenar os diversos segmentos existentes e

- otimização da alocação do espaço físico de memória através de páginas fixas.

II. ARQUITETURAS PARA PROCESSAMENTO PARALELO

A performance de um determinado sistema computacional depende de vários fatores, que podem ser agrupados em duas categorias. A primeira classe leva em consideração apenas os aspectos externos ou inerentes às máquinas que estão sendo usadas. A eficiência do software de aplicação, traduzida pela organização e qualidade dos algoritmos empregados, são exemplos de fatores situados nesta classe. Outro fator importante é a interação com o homem, que quando bem explorada resulta numa máxima utilização do potencial humano, sem que suas limitações afetem a performance do sistema como um todo.

A segunda classe de fatores diz respeito às características dos elementos utilizados para realizar o processamento. Basicamente, incluem-se neste caso o hardware do equipamento e o software que o controla. Atualmente não são bastante nítidas as fronteiras entre estes dois territórios, razão pela qual se deve sempre considerar a influência de uma decisão em ambas as partes.

Este capítulo irá discorrer sobre as principais atividades que consomem o trabalho de pesquisa da atualidade, tentanto maximizar a performance dos computadores atuais através desta última classe de alternativas.

II.1. AUMENTANDO A PERFORMANCE DOS COMPUTADORES DIGITAIS

Dando ênfase especial às características físicas da máquina, podem ser citados alguns fatores que influenciam bastante a performance de um sistema computacional.

- Tecnologia de fabricação dos componentes,
- Adequação da arquitetura à aplicação, e
- Grau de concorrência com que são executados os programas.

O primeiro item é de vital importância, porque foram os progressos introduzidos nos últimos anos que possibilitaram os

enormes aumentos de velocidade, densidade e sofisticação dos circuitos lógicos. A origem desta evolução se baseou na perspectiva de redução dos custos de processamento de componentes, proporcional ao aumento da densidade dos mesmos. Este fato, aliado à demanda de sistemas mais compactos - por razões diversas como: tamanho, confiabilidade, etc. - incentivaram ainda mais o refinamento da produção, criando nas indústrias do setor uma espécie de compulsão à miniaturização.

Os dispositivos atuais apresentam resolução da ordem de dois microns, como podemos ver na Figura II.1. A expectativa para a próxima década é reduzir este tamanho em uma ordem de grandeza, forçando uma mudança na unidade de medida para angstrons (2). A fotolitografia, o meio pelo qual os resistores químicos são expostos através de máscaras para definir a geometria dos dispositivos, é o processo que garante as menores dimensões possíveis atualmente.

Entretanto, a utilização da luz para exposição estará sempre limitada à faixa de 1 um, quando então passará a ser imperativo o uso de raios-x, feixes de elétrons ou feixes de íons no processo litográfico. É prevista também a substituição dos reveladores líquidos - que apresentam um comportamento irregular - por um processamento a seco, utilizando plasma.

Por outro lado,mesmo que os problemas de exposição e revelação das geometrias submicrométricas sejam resolvidos, não há garantia de que seja possível uma produção em massa destes componentes. Antes que os limites da tecnologia atual sejam ultrapasssados, outros fatores precisarão ser levados em consideração. Será preciso entender e modelar o comportamento das submicro-estruturas (Figura II.2), descobrir um meio de conectar a enorme quantidade de dispositivos que serão produzidos num mesmo substrato (Figura II.3) e desenvolver encapsulamentos adequados à rápida dissipação de calor (Figura II.4).

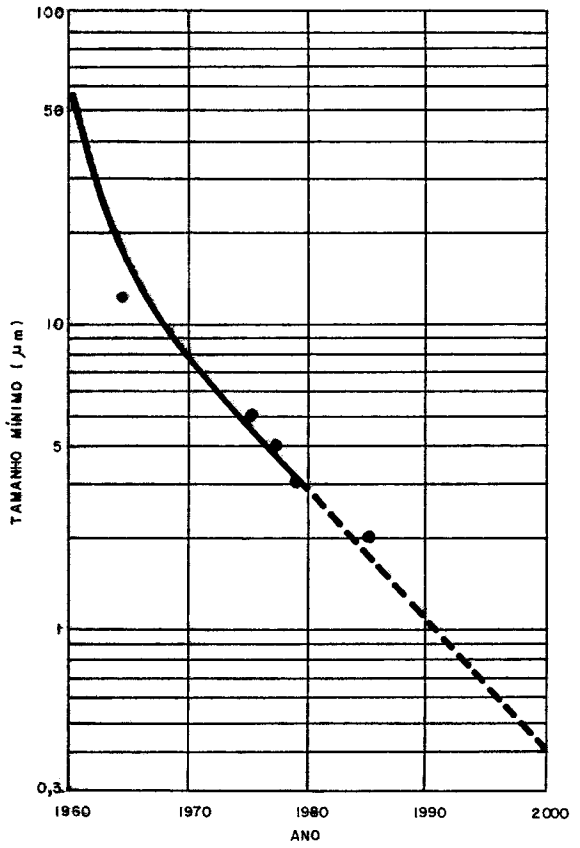


FIGURA II.1 - TAMANHOS MÍNIMOS EM CIRCUITOS INTEGRADOS: UMA REDUÇÃO VERTIGINOSA.

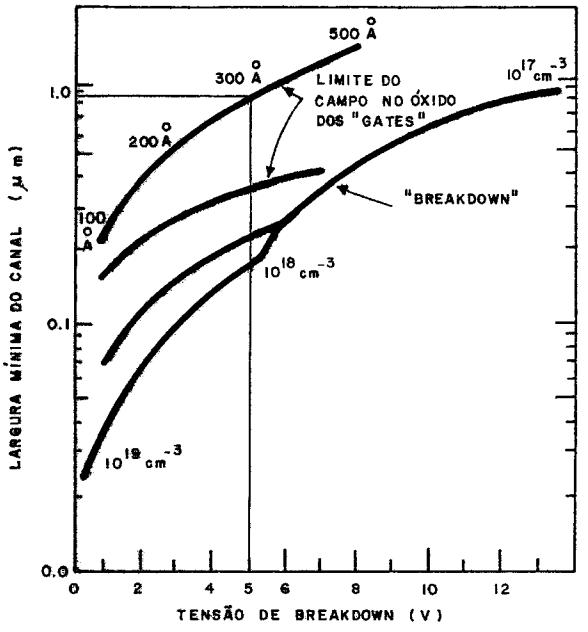
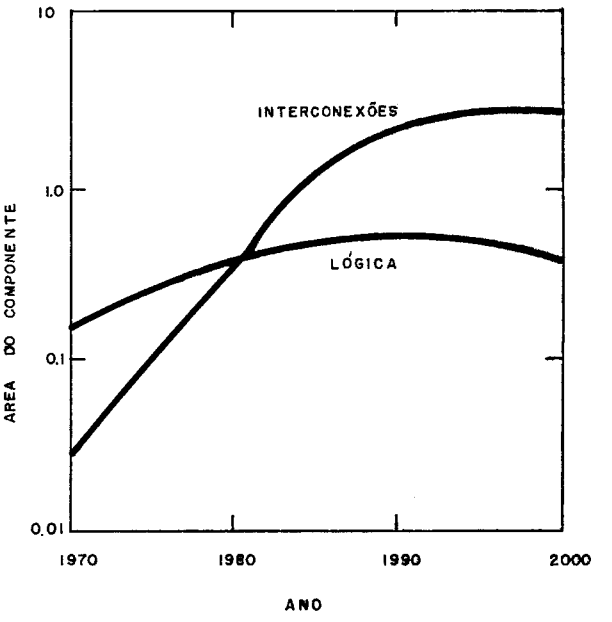


FIGURA II.2 - "BREAKDOWN": OS CAMPOS ELÉTRICOS CRESCEM À MEDIDA QUE OS DISPOSITIVOS DIMINUEM. UM "GATE" DE 300 Å DEVERÁ ROMPER-SE COM UMA TENSÃO DE 5 V, QUANDO O CANAL ULTRAPASSAR A MARCA DE 1 μm. ESPERA-SE UMA REDUÇÃO DAS FONTES DE ALIMENTAÇÃO PARA CIRCUITOS VLSI.

FIGURA II.3 - LIGAÇÕES :
AS LIGAÇÕES INTERNAS DEMANDAM CADA VEZ MAIS ESPAÇO NOS "CHIPS". ESTE NÃO É DOS PIORES PROBLEMAS, MAS VEM MERECENDO ATENÇÃO ESPECIAL DE ALGUNS FABRICANTES.



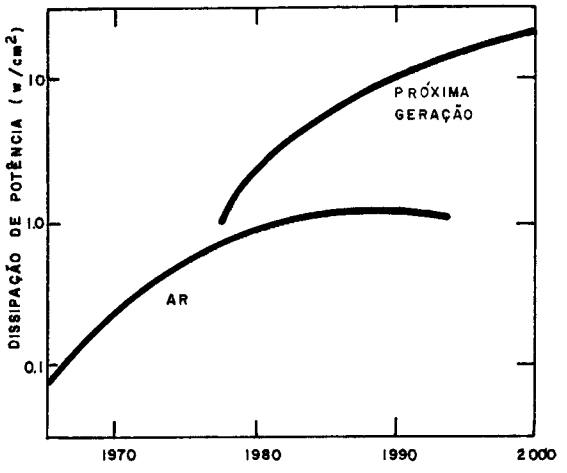


FIGURA II.4 - DISSIPAÇÃO DE CALOR :
A LIMITAÇÃO DOS INVÓLUCROS QUE UTILIZAM AR OBRIGARÁ A UTILIZAÇÃO DE UM OUTRO MEIO REFRIGERANTE (MEIOS LÍQUIDOS PERMITEM UM LIMITE DE 20 w/cm²).

As tecnologias disponíveis atualmente são várias, e a Tabela II.1 apresenta uma comparação entre as principais (2). Esta tabela não inclui algumas tecnologias "não-silicianas", como por exemplo as junções Josephson, que utilizando nióbio ou chumbo apresentam uma velocidade natural da ordem de 1ps com produto velocidade-potência no mínimo três ordens de grandeza menor que os dispositivos de Si e GaAs (3).

Os principais candidatos para aplicação em VLSI ("Very Large Scale Integration") são as tecnologias I^2L (ou outra forma de lógica-de-corrente), n-MOS, C-MOS e GaAs (2).

O futuro dos dispositivos bipolares é incerto, por causa do grande número de passos necessários ao seu processamento, principalmente porque na maioria deles é necessária a introdução de impurezas por difusão ou implantação de íons. Outro problema dos dispositivos bipolares é a tendência a dissipar mais potência que as outras tecnologias. Uma excessão à regra são os dispositivos I^2L, que com um baixo produto velocidade-potência poderão se compatibilizar com o processamento VLSI. Incluem-se nesta lista algumas variantes desta tecnologia que estão em fase de estudo: I^2L/MTL ("merged-transistor logic"), ISL ("integrated schottky logic") e STL ("schottky transistor logic").

As tecnologias MOS deverão representar um papel importante em VLSI por diversas razões. Assim, apesar da perspectiva então pessimista no aumento da densidade, os projetistas de dispositivos agiram acertadamente quando decidiram investir em melhoramentos nesta tecnologia (4). Os dispositivos MOS consomem pouca potência e são também auto-isolantes. Isto significa que as cargas podem ser armazenadas nos "gates" capacitivos, viabilizando, por exemplo, a lógica dinâmica empregada na construção de memórias.

Uma razão importante para o uso de silício na fabricação de componentes eletrônicos é a facilidade com que são criadas camadas isolantes neste material. Entretanto, a demanda de dispositivos mais rápidos e densos leva a que se continue a investigar materiais alternativos. Contudo, existem ainda

PROPRIEDADE	TECNOLOGIA ATUAL - 1979 - 1982								FUTURO, 1985 - 1990	
	BIPOLAR				MOS				SILÍCIO EM SAFIRA (sos)	ARSENIATO DE GÁLIO
	TTL	LSTTL	ECL	I^2L	p-MOS	n-MOS	C-MOS(bulk)	C-MOS(SOS)		
COMPLEXIDADE DO PROCESSO (NÚMERO DE PASSOS REALIZADOS)	18 A 22	18 A 23	19 A 23	13 A 17	8 A 14	9 A 15	14 A 17	14 A 20	14 A 20	16
COMPLEXIDADE DA LÓGICA (NÚMERO DE COMPONENTES EM UMA PORTA C/ 2 ENTRADAS)	12	12	8	3 A 4	3	3	4	4	3 A 4	2
DENSIDADE DE EMPACOTAMENTO (PORTAS/MM^2)	10 A 20	20 A 40	15 A 20	75 A 150	75 A 150	100 A 200	40 A 90	100 A 200	200 A 500	300 A 1000
TEMPO DE PROPAGAÇÃO (ns) (VALOR TÍPICO)	6 A 30 (10)	2 A 10 (5)	0.7 A 2 (2)	7 A 50 (20)	30 A 200 (100)	4 A 25 (15)	10 A 35 (20)	4 A 20 (10)	0.2 A 0.4 (0.3)	0.05 A 0.1 (0.07)
PRODUTO DE VELOCIDADE DE POTÊNCIA (pJ)	30 A 150	10 A 60	15 A 80	0.2 A 2	50 A 500	5 A 50	2 A 40	0.5 A 3 0	0.1 A 0.2	0.01 A 0.1
CHANCE DE MELHORAMENTO	BAIXA	BAIXA	BAIXA	MODERADA	BAIXA	ALTA	MODERADA	ALTA	ALTA	ALTA

TABELA II.1 - OPÇÕES TECNOLÓGICAS

problemas a enfrentar, como no caso das memórias de bolhas magnéticas que, após conseguirem superar em quatro vezes a densidade dos dispositivos de silício, tiveram sua produção interrompida por diversos fabricantes de renome internacional. Outras tecnologias, como as que usam o arseniato de gálio - que é extremamente tóxico e não permite a criação de camadas isolantes, e os materiais supercondutores das junções Josephson - que só funcionam à temperaturas perto do zero absoluto, tem atualmente custos que não incentivam o seu processamento e operação em escala comercial.

Outro fator que influencia a performance dos computadores digitais é a compatibilidade entre a sua arquitetura e as necessidades para as quais eles estão sendo projetados (5). Eis alguns parâmetros que são normalmente observados durante a especificação de uma máquina:

1. Tamanho da palavra: Pode-se medir a velocidade de um computador através do número de bits por ele processados em cada unidade de tempo. Observando-se que um único processador convencional opera uma palavra por vez, chega-se à conclusão de que a otimização do tamanho da palavra, para atender a uma determinada aplicação, é altamente desejável. Este fato pode ser notado durante a execução de programas numéricos que necessitam de precisão múltipla, em computadores com palavras de tamanho reduzido.

2. Conjunto de instruções: As aplicações que utilizam um determinado tipo de operação muito frequentemente demandam um processo especial de tratamento que dedique atenção a elas. Em caso contrário, será necessária a emulação destas operações por um conjunto de outras primitivas, o que certamente ocasionará certa ineficiência. Algumas situações críticas requerem até a introdução de coprocessadores ou controladores inteligentes, especialmente projetados para executar uma função específica. São casos típicos e bastante conhecidos os processadores de ponto flutuante e os canais que interligam dispositivos de entrada/saída diretamente à memória.

3. Organização do sistema: A topologia de interconexão dos elementos principais - unidade(s) de controle, unidade(s) aritmética(s) e lógica(s), módulo(s) de memória e controlador(es) de E/S - desempenha um papel bastante importante na caracterização de uma máquina. É através da topologia que serão especificadas as regras que irão governar o fluxo de dados e controle entre as unidades, incluindo a possibilidade de caminhos simultâneos.

4. Dispositivos de entrada/saída: Normalmente a especificação de um sistema dedica um grande esforço na parte que se refere à manipulação interna de dados. Mas a facilidade com que o computador se comunica com o resto do sistema também contribui de forma bastante significativa para sua performance final (6). Os pontos principais a serem considerados são em primeiro lugar a compatibilidade entre a demanda da troca de informação desejada e a capacidade do periférico em atendê-la. Este requisito é influenciado não só pelo tipo de unidade a ser usada, mas também pelas conexões existentes entre processador(es), memória(s) e periférico(s). Em segundo lugar deve ser escolhida uma forma de tratar a entrada/saída, (Figura II.5), que satisfaça aos requisitos de tempo de resposta, taxa de ocupação do processador, etc..

Um outro fator que exerce influência sobre a performance de um sistema computacional diz respeito ao grau de paralelismo com que são executadas as suas funções. A descentralização da inteligência permite melhorar a eficiência de uma máquina sem que seja necessário aumentar a velocidade individual de seus componentes. Contudo, para que se possa tirar o melhor proveito do paralelismo é preciso que sejam investigadas estruturas que melhor se adaptem a cada caso. As experiências que vem sendo realizadas tem comprovado o potencial desta solução. Os maiores problemas encontrados estão associados à maior complexidade funcional das estruturas paralelas, devido justamente à concorrência existentes entre suas partes.

O assunto paralelismo será abordado neste trabalho em seguida. A discussão será, a princípio, orientada para a definição e

classificação das arquiteturas para processamento paralelo.

II.2. CARACTERÍSTICAS DOS COMPUTADORES PARALELOS

A disciplina de arquitetura de computadores está atualmente em um período de transição. O advento de VLSI e de ferramentas automáticas para o desenvolvimento de sistemas (7) liberaram os projetistas de restrições que até hoje impediram uma evolução maior na área. Como consequência, vive-se hoje uma fase de consolidação de conhecimentos, através da realização e observação de experiências - pré-requisitos essenciais para a elaboração de uma boa formulação científica. O que se apresenta a seguir é um levantamento destas experiências.

II.2.1. Modelo de von-Neumann

John von Neumann estabeleceu uma arquitetura que se aplicava a um "instrumento para processamento de informações" (8). Este instrumento era composto por uma unidade central de processamento (CPU), que operava sequencialmente em dados baseados e armazenados de forma serial numa memória. Esta filosofia de execução sequencial dominou os computadores digitais por mais de trinta anos, fornecendo máquinas versáteis, gerais e com uma excelente relação custo-benefício. Os principais elementos desta arquitetura encontram-se na Figura II.6. A maioria das arquiteturas que iriam suceder à máquina de von Neumann continuariam a utilizar tais elementos, sob diferentes formas de agrupamento.

A memória consiste de células de informação ordenadas sequencialmente, que são acessadas normalmente através de endereço ou conteúdo. As operações usualmente previstas para a memória são leitura e escrita, mutuamente exclusivas.

A unidade aritmética e lógica (ALU), como o nome indica, é responsável pelas transformações matemáticas que são realizadas nos dados.

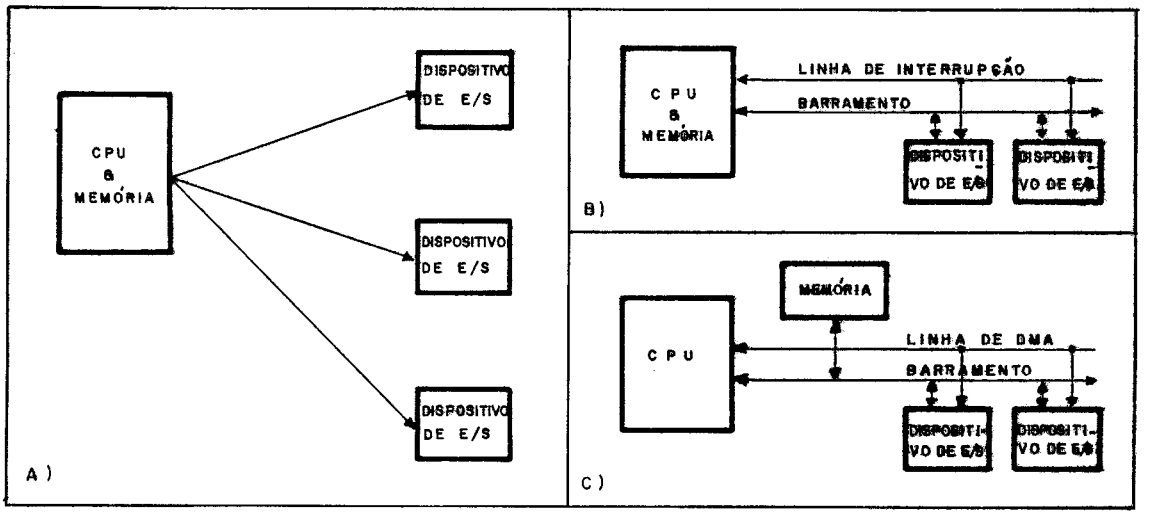


FIGURA II. 5 - CASOS TÍPICOS DE TRATAMENTO DE E/S :

A) "LOOP" DE ESPERA ("POLLING");
B) INTERRUPÇÃO E
C) ACESSO DIRETO A MEMÓRIA

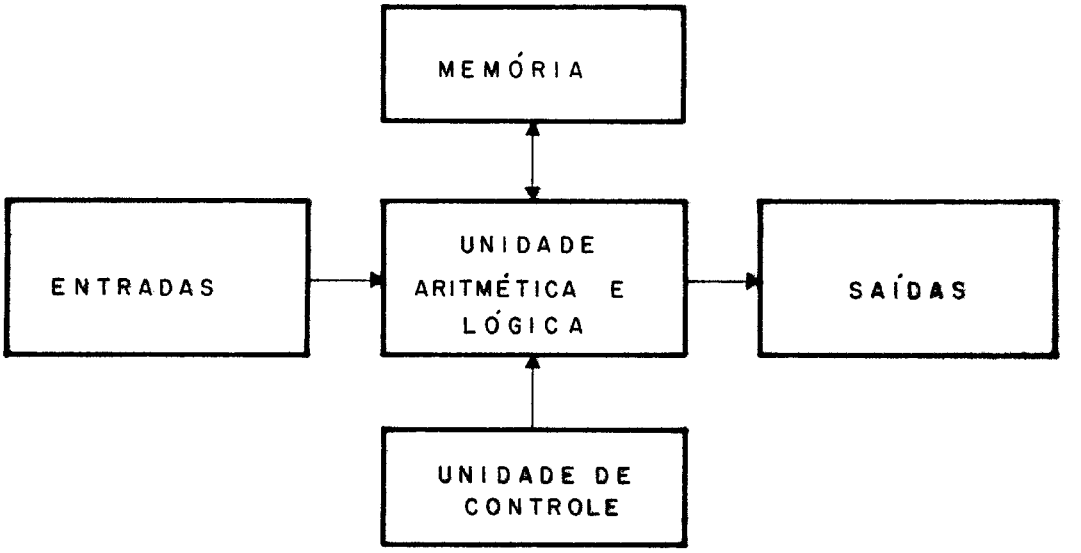


FIGURA II. 6 - ARQUITETURA PROPOSTA POR VON NEUMMAN

A unidade de controle coordena o intercâmbio de informações entre as demais unidades, bem como as operações realizadas por elas. É também esta unidade que responde aos sinais externos de interrupção através de alteração na seqüência de execução dos programas. O termo processador se refere à combinação de uma unidade de controle e uma ou mais ALU's.

As unidades de entrada e saída (E/S) possibilitam ao computador a troca de informações com outros equipamentos. As operações de E/S são utilizadas com quatro finalidades: armazenamento de informações, comunicação com o elemento humano, comunicação com outros computadores e transdução de sinais.

As linhas que interligam os módulos da Figura II.6 representam os canais de endereçamento, controle e dados do computador. A capacidade destes canais é caracterizada por vários fatores como a impedância dos circuitos de acoplamento, banda passante dos condutores, ruído, taxa de falhas desejada, quantidade de linhas, etc.. Os meios de conexão constituem um capítulo importante nas arquiteturas paralelas.

II.2.2. CLASSIFICAÇÃO DE FLYNN

Até o início da década de 70 não havia uma classificação ou codificação para as estruturas "von Neumannianas" que fosse largamente aceita. A razão é que ou o assunto era abordado sob um ponto de vista microscópico demais, através da teoria dos autômatas, ou então eram tomados como referência grupos específicos de máquinas e/ou problemas. Flynn então sugeriu uma divisão hierárquica (9) que, utilizando o conceito de "stream" (*), tratava do assunto de uma forma bastante macroscópica, sem fixar-se em nenhum ambiente particular. A classificação se baseava no grau de interação entre os "streams" de instruções e dados das diversas estruturas, resultando no aparecimento de quatro tipos de organização:

1. "Single-instruction stream/single-data stream" (SISD): Este grupo representa os equipamentos convencionais de computação, onde as instruções são executadas sequencialmente, manipulando um dado de cada vez.

2. "Single-instruction stream/multiple-data stream" (SIMD): Estes computadores apesar de executarem uma única instrução por vez, manipulam várias peças de informação simultaneamente. Incluem-se nesta categoria os "array processors" em geral e as máquinas associativas.

3. "Multiple-instruction stream/single data stream" (MISD): Há controvérsia na literatura sobre a existência ou até mesmo sobre o significado de uma máquina MISD, porque isto implicaria na manipulação simultânea de um mesmo dado por duas ou mais instruções. É óbvio que isto não é válido para uma única fração elementar de dado, mas o conceito de "stream" pode ser estendido para tipos de dados mais complexos, passando então a fazer sentido esta classificação. O processador "pipeline" satisfaz, segundo alguns, os critérios necessários aos membros desta classe.

4. "Multiple-instruction stream/multiple -data stream" (MIMD): Esta categoria abrange todos os sistemas que, executando mais de uma instrução por vez, manipulam simultaneamente várias peças de dados. A revolução dos mini e micro-computadores permitiu o desenvolvimento de todo o potencial das organizações múltiplas, tanto em rede de computadores como em arquiteturas a multiprocessador.

(*) "Stream" neste contexto significa uma sequência de itens (instruções ou dados) na forma em que são executados ou operados por um processador.

II.2.3. OUTROS MODELOS

Para complementar a classificação dos computadores paralelos é preciso fugir do conceito de máquina orientada para "streams" (de dados ou instruções) e até mesmo do modelo de von Neumann. Para isso, faz-se necessária uma abstração dos conceitos de execução sequencial de programas e de memórias endereçadas globalmente, que vinculam fatalmente uma arquitetura aos casos já citados anteriormente. O trabalho de classificação do grupo não alinhado à von Neumann ainda está para ser feito, havendo no entanto alguns casos isolados que podem ser citados. É o caso de máquinas digitais (10 a 15) especialmente projetadas para uma determinada aplicação e dos sistemas de fluxo de dados (23 a 30) que vem sendo estudado como uma alternativa geral para as máquinas e linguagens tradicionais.

A Figura II.7 engloba as categorias citadas até o presente momento. Os demais itens deste capítulo irão dissertar sobre algumas arquiteturas não convencionais que, através da exploração do paralelismo, tem apresentado alternativas promissoras para o desenvolvimento de sistemas mais poderosos. O próximo capítulo é dedicado aos sistemas com múltiplos processadores. Este grupo tem a vantagem de atingir um grande espectro de aplicações, em virtude do seu baixo custo, aliado à alta disponibilidade do sistema final. Em contrapartida, a demanda de um maior volume de software através de hardware menos eficiente (em relação às outras arquiteturas) vem comprometendo uma utilização comercial mais ampla de tais sistemas.

II.3. ARQUITETURAS SISTÓLICAS

Na produção de uma máquina, devem ser levados em consideração dois tipos de custos: os não recorrentes e os recorrentes. Despesas de projeto, por exemplo, são contabilizadas de uma única vez e diluídas por toda a produção, enquanto que gastos materiais são levados em consideração em cada unidade fabricada.

As principais restrições feitas aos sistemas especiais tem

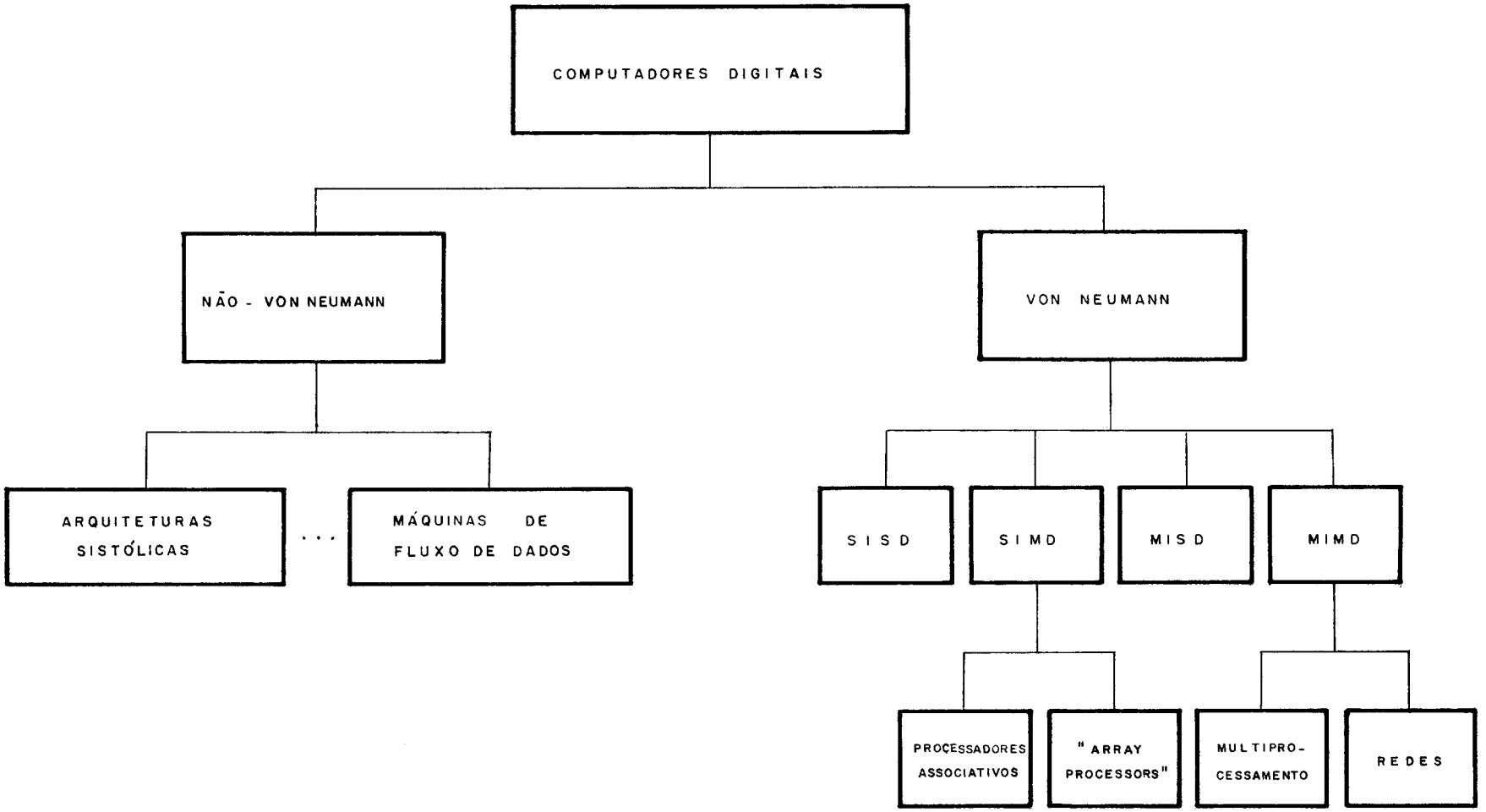


FIGURA II.7 - UMA CLASSIFICAÇÃO PARA OS COMPUTADORES DIGITAIS.

relação com o seu custo de projeto, em virtude da aplicabilidade limitada inerente a tais equipamentos.

Um dos meios para reduzir estes custos é a criação de uma metodologia que permita uma certa automatização de procedimentos. Uma idéia é decompor a estrutura que se deve projetar num agrupamento de módulos simples não muito diferentes e repetitivos. Para que a idéia seja viável técnica e economicamente é desejável que o acoplamento entre estes módulos seja bastante elementar.

As chances de sucesso nesta área aumentam consideravelmente se forem utilizados projetos em VLSI, onde um "single-chip" pode armazenar centenas de milhares de componentes. Neste nível de complexidade, torna-se imprescindível a utilização de módulos simples e regulares, similares àqueles empregados na construção de sistemas bem estruturados de software (16). Um exemplo bastante interessante desta categoria são as arquiteturas sistólicas, descritas a seguir.

As arquiteturas sistólicas (*) representam um caso típico de máquinas especiais, que foram originalmente concebidas para implementação em VLSI de operações em matrizes (17). Um sistema sistólico é composto por um conjunto de células interconectadas, cada qual capaz de realizar alguma operação simples. As vantagens registradas no projeto e implementação de estruturas simples e regulares levaram à adoção de topologias sistólicas do tipo matriz ou árvore (Figura II.8).

Nos sistemas sistólicos, a informação flui entre as células de uma maneira análoga ao sangue nas veias do corpo humano. A comunicação com o mundo exterior fica a cargo das células fronteiriças, que injetam informações no sistema de forma pulsante, como um coração.

(*) Sistólica: Adj. relativo a, ou próprio de sístole; 1. Med. Estado de contração das fibras musculares do coração Cf. diástole.

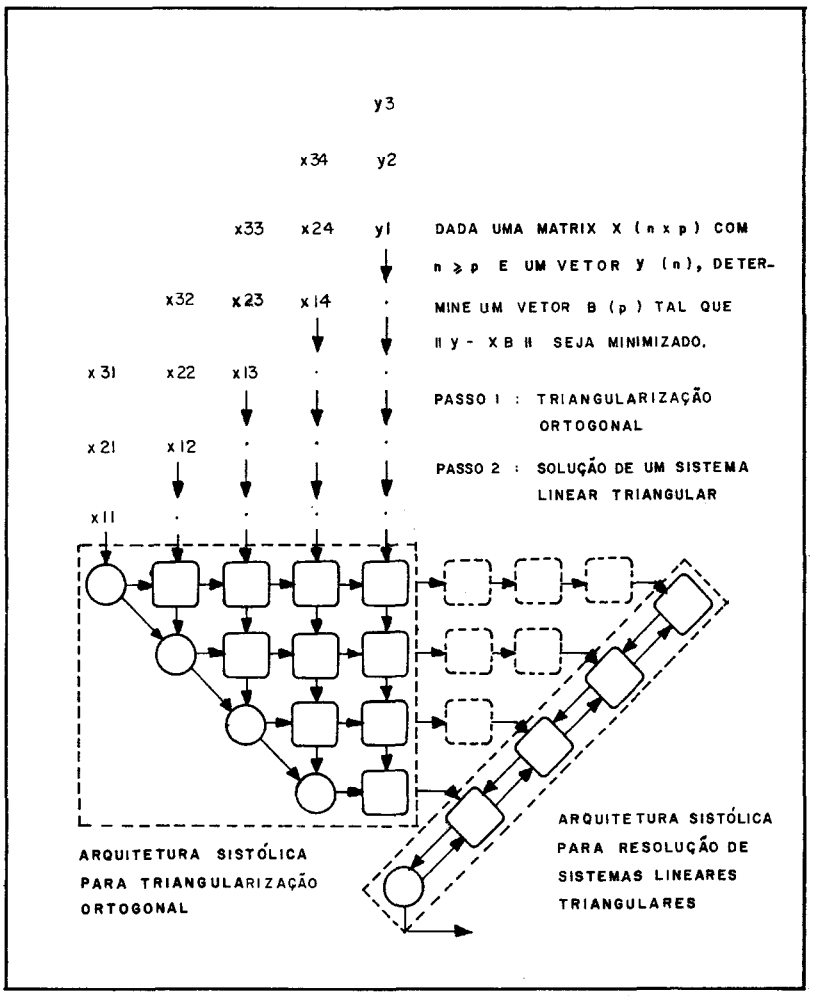


FIGURA II. 8 - SOLUÇÃO PELO MÉTODO DOS MÍNIMOS QUADRADOS UTILIZANDO MATRIZES SISTÓLICAS UNI E BI - DIMENSIONAIS (p = 4)

As tarefas computacionais podem ser conceitualmente classificadas em "io-bound" ou "compute-bound", de acordo com a quantidade de operações de cada tipo que são realizadas. Por exemplo, o algoritmo de multiplicação de matrizes representa uma tarefa "compute-bound", porque cada elemento da matriz é multiplicado por todos os elementos de alguma linha ou coluna da outra matriz. Por outro lado, a soma de matrizes é "io-bound", já que o número total de somas realizadas não é maior que o número total de elementos das duas matrizes. A partir destas considerações Kung (12) conclui que:

1. Qualquer tentativa para acelerar uma tarefa "io-bound" tem que ser baseada em um aumento da banda passante da memória. Isto pode ser conseguido com o uso de componentes mais rápidos (e caros) ou memórias intercaladas (com uma gerência mais complicada).

2. A aceleração de tarefas "compute-bound" podem, de um modo geral, ser realizadas de uma forma simples e barata através da solução sistólica.

Como exemplos, serão apresentados a seguir alguns exemplos de estruturas sistólicas (12), para resolver o problema da convolução:

Dada a sequência de pesos $w1, w2, \ldots, wk$
e a sequência de entradas $x1, x2, \ldots, xn$
computar a sequência de saídas $y1, y2, \ldots, yn+1-k$
definida por $yi = w1\ xi + w2\ xi+1 + \ldots + wk\ xi+k-1$

Este problema é representativo de uma grande classe de computações que podem ser resolvidos via projetos sistólicos. Na essência, trata-se de combinar duas sequências de uma maneira tal que resulte numa sequência de yi s. Rotinas com cálculos semelhantes podem ser encontrados em filtragem, reconhecimento de padrões, correlação, interpolação, avaliação polinomial (incluindo transformadas de Fourier discretas) e multiplicação e divisão de polinômios.

A convolução é um problema "compute-bound" por natureza, dado que cada entrada "xi" deve ser multiplicada por cada um dos "k" pesos. Se a cada multiplicação precisarmos acessar o "xi" na memória, certamente haverá um estrangulamento. As ilustrações a seguir serão simplificadas para o caso em que k = 3.

Matrizes Semi-Sistólicas com Dados Globais: Uma das técnicas para aproveitamento máximo da sequência de xi's é fazer com que cada elemento seja totalmente utilizado em todas as células de uma só vez. Dá-se o nome de "broadcasting" a esta técnica, adotada na arquitetura da Figura II.9. O princípio básico deste projeto foi inicialmente proposto para implementação de um processador de reconhecimento de padrões (18) e em multiplicação de polinômios (19). Notar que a arquitetura é capacitada para armazenamento de informações, o que minimiza o acesso à memória principal.

Matrizes Puramente Sistólicas: Embora o "broadcasting" resolva teoricamente o problema, ele apresenta dificuldades de implementação prática. A transmissão do(s) dado(s) global(is) requer o uso de uma rede que percorra todas as células, o que certamente conduzirá a problemas de expansão e/ou aceleração do sistema. O diagrama da Figura II.10 resolve os três problemas: o da convolução, o de máxima utilização das sequências e o da viabilização prática em larga escala.

Esta estrutura pode ser naturalmente estendida para realizar filtragem recursiva (20,21) e divisão polinomial (22). Outra característica desta matriz sistólica é o tempo de resposta constante, com um valor de "yi" a cada dois ciclos de tempo. Além destas duas soluções, várias outras são apresentadas por Kung (12).

Os problemas de contenção introduzidos pela excessiva quantidade de acessos à memória nos sistemas convencionais refletem uma inadequação da arquitetura ao problema, de acordo com a discussão introduzida no item II.1.

Em geral os projetos sistólicos podem ser aplicados a qualquer

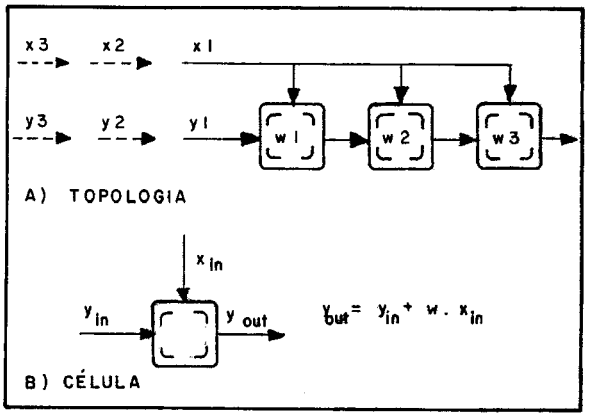


FIGURA II.9 -

SOLUÇÃO SISTÓLICA PARA A CONVOLUÇÃO, ONDE ELEMENTOS xi's SÃO FIXOS E yi's SE MOVEM SISTOLICAMENTE.

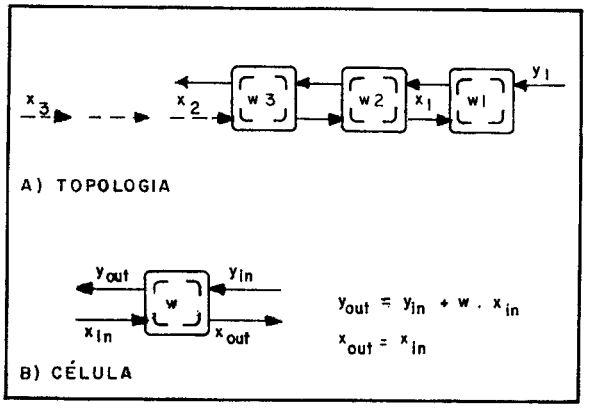


FIGURA II. 10.

SOLUÇÃO PURAMENTE SISTÓLICA, ONDE OS PESOS wi PERMANECEM ESTÁTICOS, ENQUANTO AS SEQÜÊNCIAS xi E yi MOVEM-SE SISTOLICAMENTE, EM SENTIDOS OPOSTOS.

problema "compute-bound" regular, isto é, aquele que realiza computações repetitivas em um grande conjunto de dados. O desenvolvimento de arquiteturas sistólicas irá permitir a ampliação de sistemas especiais, que apresentam rotinas bem delineadas. Mas o objetivo final é aplicar estas técnicas em ambientes genéricos de computação. Para isso são necessários investimentos na integração de componentes para tais sistemas e na criação de blocos programáveis que possam ser flexivelmente configurados em células básicas diferentes.

II.4. SISTEMAS DE FLUXO DE DADOS

A idéia de produzir uma máquina baseada em fluxo de dados data da década de 60 (23 a 25), mas só recentemente é que a pesquisa nesta área vem registrando avanços consideráveis. Alguns pesquisadores tem encarado estes computadores como sucessores em potencial das máquinas e linguagens de von Neumann (26).

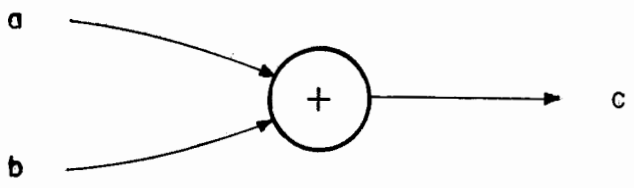
O conceito de fluxo de dados difere do mecanismo de execução sequencial de instruções sendo baseado em dois princípios:

1. Assincronismo: todas as operações são realizadas quando e somente quando seus operandos estiverem disponíveis.

2. Funcionalidade: todas as operações são expressadas através de funções, ou seja, não são considerados os endereços das variáveis mas sim os seus valores.

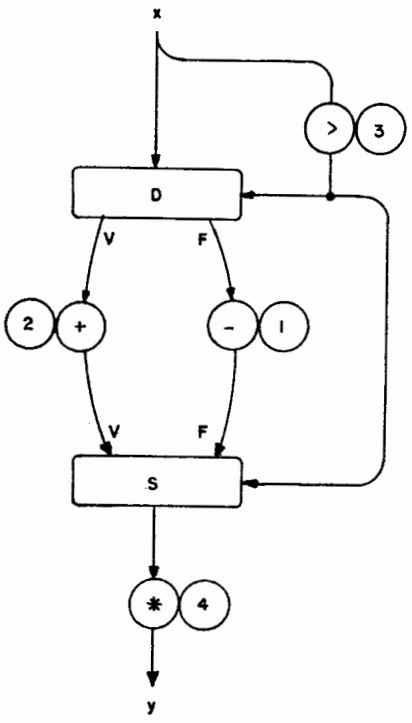
Esta filosofia reflete um mecanismo que pode ser associado a um grafo direcionado, onde cada nodo representa uma operação a ser realizada.

Os arcos "a" e "b" do operador "+" da Figura II.11 transportam os "tokens" de entrada que serão operados (somados no caso) e colocados no arco de saída "c". No caso geral, os dados fluem através do grafo, percorrendo os diversos operadores. O segundo princípio acima citado tem como consequência a eliminação dos efeitos colaterais ("side effects") do sistema, acarretando uma



c = a + b

FIGURA II. 11 - UM OPERADOR EM "DATA FLOW"



y = (if x>3 then x + 2 else x - 1) * 4

FIGURA II. 12 - REPRESENTAÇÃO DE UMA EXPRESSÃO EM "DATA FLOW".

total liberdade para que quaisquer duas operações sejam executadas concorrentemente.

Não é possível declarar um nodo como executável apenas pela validade do princípio número um, porque os tokens de entrada podem pertencer a diferentes partes da computação. Gajski et al (27) apresentam cinco soluções para este problema. Uma delas consiste em limitar (em um) o número de tokens residentes nos arcos a cada instante. Obtém-se daí uma possível "regra de disparo" do operador, que o torna executável somente quando os tokens de entrada estiverem disponíveis e o de saída desocupado.

As estruturas para realizar testes de condição e iterações estão ilustradas através de exemplos nas Figuras II.12 e II.13 respectivamente (28). O nodo distribuidor (D) envia o dado de entrada para a saída "V" ou "F" dependendo do estado da entrada booleana de controle. Analogamente, o nodo seletor (S) transfere para a saída uma das duas entradas, de acordo com a porta de controle.

Como se pode observar nos exemplos anteriores, é bastante direta a relação entre os grafos direcionados (2º) e a arquitetura de fluxo de dados propriamente dita. Contudo, este não é um meio muito apropriado de programação em virtude da dificuldade de manipulação de diagramas e da alta propensão a erros. Vários pesquisadores tem orientado seus trabalhos para o desenvolvimento de formas de representação de programas que se adaptem bem aos conceitos próprios deste tipo de computação (30). Três classes de linguagem de alto nível tem sido consideradas (27): a primeira é a classe imperativa, que tenta adaptar linguagem como FORTRAN e PL-I para o processamento de fluxo de dados. A segunda é a classe funcional, representada por linguagens que apresentam compatibilidade natural, como o Lisp e o FP que são imunes a efeitos colaterais. A terceira classe é composta por linguagems especialmente projetadas, denominadas linguagens para fluxo de dados. As mais conhecidas são Id, LAU e Val, que reunem características das linguagens imperativas e das linguagens funcionais. Estas características são discutidas por Ackerman (30), e as principais são a semântica funcional - que

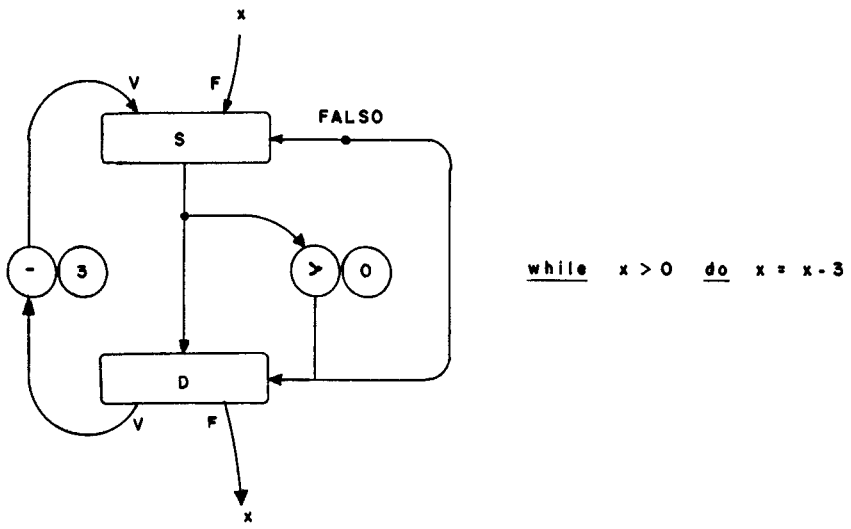


FIGURA II. 13 - REPRESENTAÇÃO DE UMA ITERAÇÃO EM " DATA FLOW "

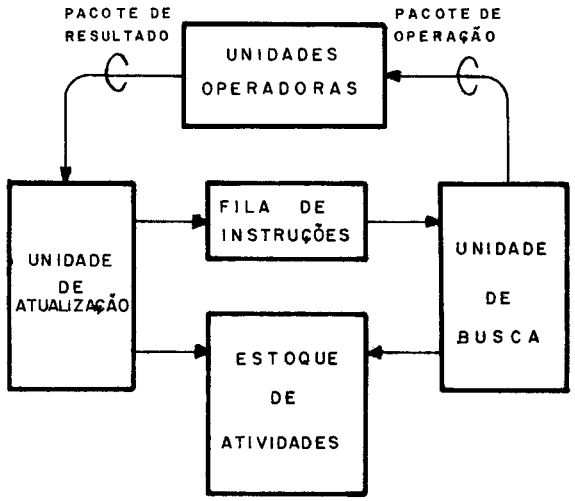


FIGURA II. 14 - MECANISMO BÁSICO PARA EXECUÇÃO DE INSTRUÇÕES NA MÁQUINA DE FLUXO DE DADOS.

permite ao programador abster-se do controle sobre a alocação de memória para código e dados, e o paralelismo implícito - já que o compilador é o responsável pela deteção e criação de caminhos paralelos nos programas.

O mecanismo básico para execução de instruções utilizado em diversos projetos de fluxo de dados está representado esquematicamente na Figura II.14 (5). O estoque de atividades é composto por um grupo de elementos que definem o programa que está sendo executado. Cada elemento contém os valores dos argumentos de entrada, a descrição do operador que vai ser utilizado e a definição das outras atividades elementares que deverão receber o resultado da operação. Cada atividade elementar tem um endereço de identificação, que é colocado na fila de instruções quando a "regra de disparo" do operador assim o permitir. A unidade de busca pega o endereço de uma instrução na fila e forma o pacote de operação com os dados lidos na atividade elementar correspondente. Após o término da execução da instrução pelas unidades operadoras é gerado um pacote com os resultados. A partir deste pacote, a unidade de atualização preenche os argumentos nas atividades elementares destino. É também a unidade de atualização que identifica se uma atividade elementar já atendeu a todos os requisitos para ser executada, quando então coloca o seu endereço na fila de instruções. O grau de concorrência em um determinado instante pode ser medido pelo número de entradas da fila de instruções. Este mecanismo é conhecido como pipeline circular.

As principais críticas que são feitas atualmente aos grupos que investigam este assunto estão relacionadas com a falta de compromisso com o real ambiente de computação existente atualmente. A principal razão disto é a orientação "bottom-up" que está sendo dada às pesquisas, visto que somente ao nível de operadores escalares é que foram atingidos objetivos importantes. Outro aspecto que trará dificuldade à implantação deste tipo de processamento é o conservadorismo que tende a existir nos meios computacionais. Grande esforço está sendo feito hoje em dia para padronização e aumento da produtividade do elemento humano, o que parece conflitar com as ferramentas

até agora desenvolvidas para fluxo de dados.

II.5. PROCESSAMENTO "PIPELINE"

Há bastante controvérsia na literatura sobre a classificação deste tipo de arquitetura. Flynn considerou o processador "pipeline" como um dos três tipos básicos de máquinas SIMD (9), no mesmo artigo em que introduziu tal classificação para as arquiteturas de computadores. Outras interpretações contudo citam tal arranjo como pertencente à categoria MISD (5 e 31). No escopo deste trabalho estas arquiteturas não serão posicionadas em nenhum grupo específico. Em vez disso, as técnicas de "pipelining" serão consideradas como uma filosofia de projeto, visando a obtenção de um maior aproveitamento dos componentes de hardware. Esta decisão é baseada na constatação de que esta "filosofia" é utilizada de forma bastante diversificada em aplicações que variam desde supercomputadores até unidades periféricas inteligentes, ou até mesmo na arquitetura interna de alguns microprocessadores.

Uma estrutura "pipeline" é caracterizada pela organização em série de seus elementos. Cada um deles é incumbido de operar de forma independente o dado que recebe e de transmitir o resultado para o elemento seguinte. A Figura II.15 apresenta esquematicamente uma solução para processar uma determinada função F. A subdivisão em sucessivas sub-funções fi transforma a função original em:

$$F(E) = fn\ (fn\text{-}1(...fi(...f2(f1(E)))...)...))$$

O tempo total de execução da função F é igual à soma dos tempos gastos por cada fi. Este valor permite definir o fator de latência (L) da estrutura, medido em unidades de tempo Δ t. A performance do sistema ao ser alimentado por uma entrada E é medida pela quantidade de dados processados por unidade de tempo:

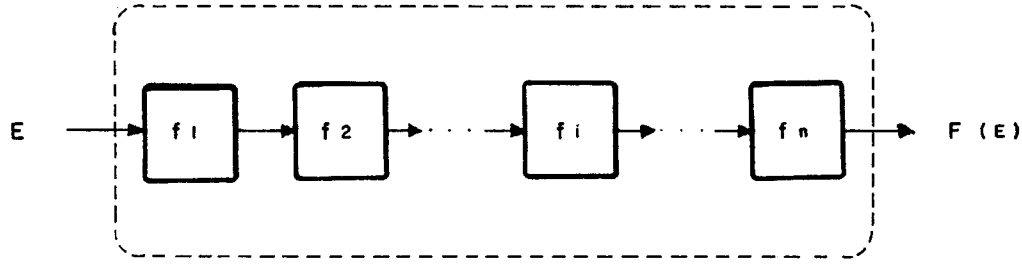
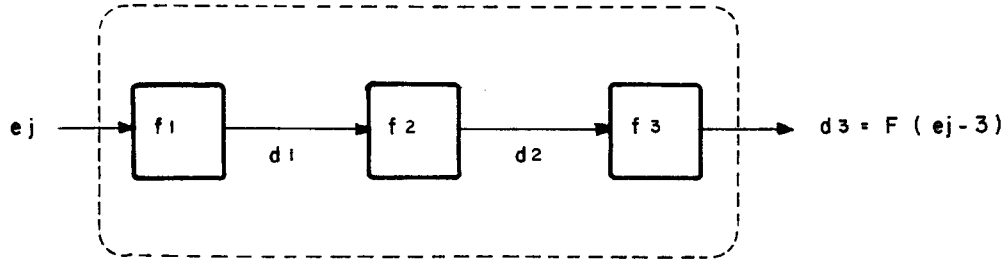


FIGURA II. 15 - ESTRUTURA "PIPELINE", ONDE CADA ELEMENTO REALIZA UMA FUNÇÃO fi SOBRE O DADO QUE RECEBE, TRANSMITINDO O RESULTADO AO ELEMENTO SEGUINTE.



$$d1 = f1\ (ej - 1)$$

$$d2 = f2\ (f1\ (ej - 2))$$

$$d3 = f3\ (f2\ (f1\ (ej-3))) = F\ (ej - 3)$$

FIGURA II. 16 - SISTEMA "PIPELINE" ALIMENTADO POR UMA SEQÜÊNCIA DE ENTRADAS ej.

$$Perf = \frac{1}{L\Delta t}$$

Admitindo-se agora que a entrada E é composta por uma sequência definida por:

E = e1, e2, ..., ej, ...

e que as funções fi são independentes, pode-se alimentar o sistema com a entrada e2 assim que f1 terminar o tratamento de e1. A aplicação sucessiva deste procedimento resultará no preenchimento da série de fi's, ficando cada elemento responsável pela geração de um dado di. A Figura II.16 representa tal situação para três elementos.

Observar que, se for mantido um fluxo de entrada constante, o sistema poderá teoricamente responder com uma saída para cada entrada injetada. A performance final foi aumentada n vezes, apesar do período de latência para tratamento de uma entrada ter continuado o mesmo.

De uma forma mais geral, é definido o fator de inércia do sistema como sendo o número de dados que estão sendo processados durante o tempo de latência para tratamento de uma entrada. A Figura II.17 ilustra este caso.

Consequentemente a expressão da performance máxima alcançada pelo sistema pipeline é dada por:

$$Perfmax = \frac{J}{L\Delta t}$$

Entretanto, a performance máxima do sistema só é obtida no caso em que a "linha de produção" está completamente cheia, ou seja, todos os operadores contém dados nas suas entradas. No caso real é necessário considerar dois fatores que contribuem para o seu esvaziamento. O primeiro está relacionado com a dificuldade de programar as sequências E de entrada de forma a manter um fluxo constante de dados. Esta preocupação tem levado os projetistas a

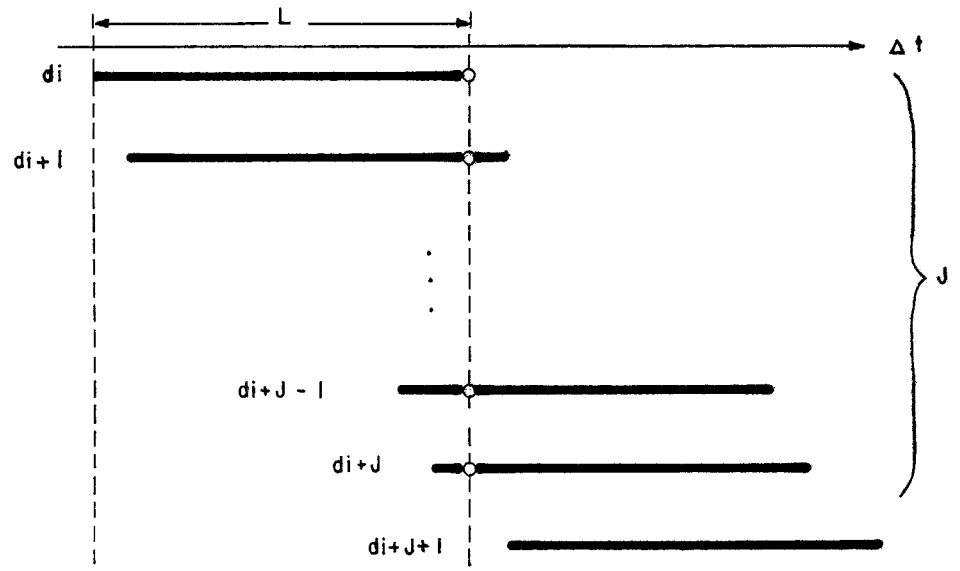


FIGURA II. 17 - O FATOR DE INÉRCIA J REPRESENTA O NÚMERO DE DADOS QUE ESTÃO SENDO OPERADOS DURANTE O TEMPO DE LATÊNCIA L Δt.

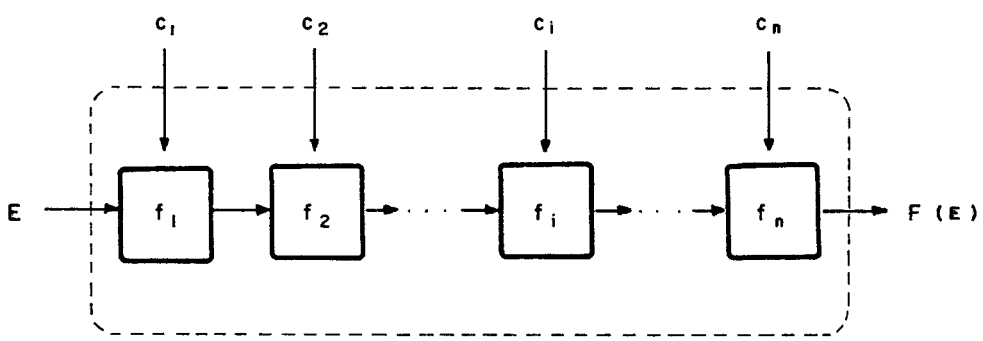


FIGURA II. 18 - SISTEMA " PIPELINE " COM ENTRADAS EXTERNAS Ci.

desevolver ferramentas de software especiais que permitam arrumar o programa de forma a extrair o máximo possível de paralelismo das aplicações. Dá-se o nome de "vetorização" a essa técnica.

Outro fator que pode causar retardo na sequência de processamento normal do sistema está relacionado à disponibilidade de dados externos. Para exemplificar, considere um sistema "pipeline" como o da Figura II.18. Neste sistema, a função de cada elemento não depende somente do dado de entrada di, mas também de um outro dado externo ci. É óbvio que qualquer retardo introduzido pela ausência dos dados externos irá se refletir na performance final do sistema. Flynn (9) analisa a situação em que o dado ci é dependente das próprias funções da estrutura, ou seja:

ci = h (d1, d2, ..., dj, ..., di-1)

onde cada dj representa o dado gerado pelo elemento j da linha de "pipes".

Outras características interessantes dos sistemas pipeline surgem quando o custo é levado em consideração. Sterling (5) demonstra para um caso simples que é grande o custo adicional de adaptação para processamento pipeline. Em alguns casos, apesar do ganho em performance há significativa perda na relação performance/custo. Na mesma referência é investigada também a redução da eficiência causada pelos incrementos de hardware que são necessários para garantir o perfeito funcionamento do sistema "pipeline".

Os problemas de perda de eficiência em sistemas que utilizam processamento "pipeline" podem ser observados através de medidas de performance, em exemplos reais de aplicação. O estudo de Bucy, citado em (32), escolheu um problema de filtragem não linear que requeria aproximadamente 83000 operações em ponto flutuante para cada iteração. Este problema foi resolvido em vários sistemas diferentes e a Tabela II.2 mostra a grande diferença entre as velocidades máximas teóricas e as velocidades

MÁQUINA		VELOCIDADE (M. FLOPS)			CUSTO APROXIMADO (US DOLAR / FLOP)
NOME	TIPO	MÁXIMA TEÓRICA	REAL		
			ABSOLUTA	RELATIVA AO MAX.	
CRAY - 1	SUPERCOMPUTADOR	140	38.4	27 %	0.21
STAR - 100	SUPERCOMPUTADOR	50	16.8	34 %	0.48
AP - 120 B	PROCESSADOR DE MATRIZES	12	5.9	49 %	0.03

TABELA II 2 - ALGUNS DADOS DO ESTUDO DE BUCY SOBRE A PERFORMANCE DE SISTEMAS QUE UTILIZAM PROCESSAMENTO PIPELINE.

médias reais alcançadas por estas máquinas. A medida de velocidade utilizada é o Megaflop (mflop), que é equivalente a um milhão de operações em ponto flutuante por segundo.

II.6. PROCESSAMENTO ASSOCIATIVO

Os primeiros estudos envolvendo o conceito de processamento associativo datam da década de 50 (33), embora só há dez anos tenham sido registradas realizações práticas significativas de tais arquiteturas (34). A motivação para o desenvolvimento desta área partiu da constatação de que as soluções para processamento de uma certa classe de problemas demandavam excessivo tempo de execução nas máquinas convencionais. Não se tratava apenas de um simples aumento de velocidade, mas sim de uma utilização mais racional das potencialidades disponíveis.

Um computador associativo típico, representado esquematicamente na Figura II.19 compreende: uma memória associativa para armazenar os dados; uma unidade de controle para coordenar a execução dos programas; uma memória de instruções e unidades de E/S.

A memória associativa é um dispositivo armazenador de informações organizado por células, que são acessadas por intermédio de seu conteúdo. Além disso, as células da memória associativa são capazes de realizar o processamento nos dados localmente, eliminando assim a necessidade de transferências para alguma unidade central. A consequência disso é a fusão das atribuições da memória e da unidade aritmética e lógica em um único módulo. Uma outra característica dos processadores associativos é a separação física da memória em duas partes: código e dados.

De acordo com a sua organização, podemos classificar um dispositivo associativo em quatro categorias: totalmente paralelo, bit-série, palavra-série e orientado por blocos.

Os dispositivos totalmente paralelos podem por sua vez serem

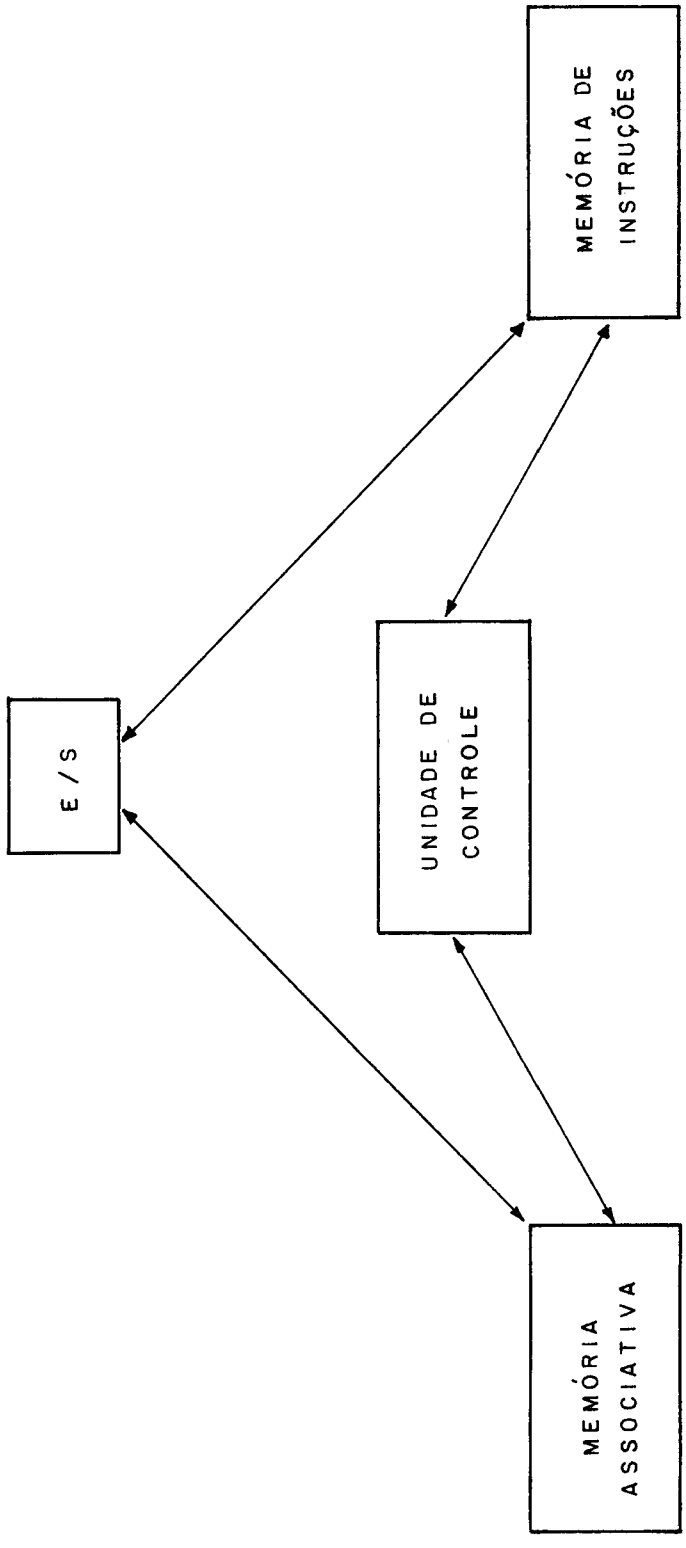


FIGURA II. 19 - ESTRUTURA DE UM COMPUTADOR ASSOCIATIVO TÍPICO.

divididos em sistemas organizados por palavra e com lógica distribuída. Nos sistemas totalmente paralelos organizados por palavra as operações são utilizadas simultaneamente em todas as palavras da memória. Este grupo representa o processador associativo ideal no que diz respeito à velocidade e versatilidade. Porém a grande quantidade de circuitos necessários à esta arquitetura não incentiva muito a sua implantação. O segundo grupo de processadores associativos puramente paralelos é o de lógica distribuída. Esta organização é bastante eficiente na manipulação de dados de tamanho variável. A memória de lógica distribuída é composta por células estanques de armazenamento, onde a comunicação é feita entre duas células vizinhas e a unidade de controle. Um processador associativo deste tipo foi desenvolvido pelo "Bell Laboratories" para a "U.S.Army Advanced Ballistic Missile Defense Agency", sendo batizado de "Parallel Processing Ensemble" (PEPE) (35). PEPE era usado em conjunto com um computador sequencial de alta velocidade.

A categoria de dispositivos associativos bit-série representam um compromisso entre os aspectos funcionais dos sistemas totalmente paralelos e os custos envolvidos na implementação de tais dispositivos. Em vez de dedicar inteligência a cada célula (bit) da memória, é reservada uma única unidade lógica para cada palavra. Assim sendo, somente um bit de cada palavra é processado simultaneamente em toda a memória. A desvantagem desta solução é o retardo introduzido no processamento igual à quantidade de bits por palavra de memória. Porém, além da grande redução ainda registrada no tempo total de processamento, há a vantagem de uma substancial economia de hardware em relação aos sistemas totalmente paralelos. Um modelo deste grupo é o sistema STARAN (36) fabricado pela Goodyear Aerospace Corp..

Os computadores que processam as palavras serialmente não utilizam memórias endereçadas por conteúdo, não podendo ser rigorosamente considerados como dispositivos associativos. O que diferencia estas máquinas dos computadores convencionais é o hardware especial que controla o processamento durante a varredura de um trecho da memória sequencial. A utilização de

uma única instrução para realizar este processamento economiza ao computador o tempo de "fetch" repetitivo de instruções. Operações deste tipo são suportadas por diversos computadores atualmente, como o VAX 11/780 e o microprocessador INTEL 8086.

Para aplicações que requeriam uma grande capacidade de armazenamento, as soluções associativas até agora apresentadas não são viáveis. Os sistemas bit-série tornam-se muito caros e os sistemas palavra-série resultam num tempo de processamento excessivo. Algumas tentativas para estender o conceito de associatividade às memórias de massa (37 e 38), procuraram tirar vantagem dos discos de cabeça fixa de alta performance. As propostas sugerem a associação de tais equipamentos a uma lógica externa, que tenha capacidade de manipular os dados à medida que eles passem sob a cabeça de leitura.

A programação dos dispositivos associativos é quase toda feita em baixo nível, quer seja em linguagem de máquina ou microcódigo. O sistema STARAN, por exemplo, utiliza um montador simbólico (APPLE - "Associative Processor Programming Language"), que sofre adaptações especiais em cada ambiente que é instalado. Uma outra alternativa, que permite o desenvolvimento de programas através de linguagens de alto nível é a inclusão de construtos específicos para processamento associativo em linguagens já existentes. Convém ressaltar que há normalmente uma dificuldade natural com a programação de máquinas associativas, em virtude das diferenças conceituais inerentes aos sistemas SIMD.

As aplicações mais adequadas ao processamento associativo são aquelas que apresentam uma ou mais das seguintes propriedades (39): busca rápida em grandes bancos de dados; realização de operações aritméticas e lógicas em grandes conjuntos de dados ou manipulação de bancos de dados que sofram alterações dinâmicas.

Estas aplicações incluem soluções para problemas de processamento de sinais, armazenamento e recuperação de informações, suporte para sistemas de computadores além de aplicações numéricas tais como otimização de funções, solução de

equações diferenciais, operações com matrizes, manipulação de grafos, etc..

A primeira experiência prática com um sistema associativo real de proporções significativas (34) foi dedicada ao processamento de sinais de radar. A Administração Federal da Aviação dos Estados Unidos (FAA) utilizou-se de um sistema para controle de tráfego aéreo, baseado na máquina associativa de Knoxville para monitorar aeronaves, isolando as que estivessem em rotas de colisão ou situações potencialmente perigosas. A performance alcançada por tal sistema excedeu em duzentas vezes o tempo de resposta das máquinas convencionais da época. O sistema STARAN, já citado, foi construído a partir desta experiência.

Nos sistemas de gerenciamento de bancos de dados o formato, estrutura e quantidade de informações variam durante o tempo de vida útil do sistema, o que torna adequada a sua implementação através de dispositivos associativos (40). Apesar das dificuldades de construção prática de uma grande memória endereçada por conteúdo, prosseguem as investigações visando o desenvolvimento de memórias de massa associativas (37 e 38). Além disso, há trabalhos orientados para processamento "on-line" e compressão/descompressão de dados (41).

Algumas funções de controle presentes nos computadores podem se beneficiar dos dispositivos associativos. É o caso por exemplo de pequenas unidades auxiliares destinadas a controlar a alocação de memória virtual em grandes computadores (42). Os dispositivos associativos são também considerados potencialmente úteis pelos grupos interessados em processamento de linguagens.

Os processadores associativos são propostos como sendo sistemas computacionais bastante flexíveis para processamento de informações não-numéricas. Tais sistemas podem manter uma alta eficiência de hardware, na maioria dos casos, sem que isto incorra em um software complexo (41). São ainda necessárias pesquisas principalmente na área de arquitetura, para avaliar o potencial das diversas organizações e prover adaptações às técnicas de VLSI. Na área de software é preciso que haja

desenvolvimento de novos algoritmos para suportar as estruturas de dados mais comuns (árvores, listas, pilhas, etc.), de linguagem de alto nível adequadas e de ferramentas de suporte.

Por outro lado, a grande confiança depositada atualmente nas linhas mais conservadoras de projetos de hardware e software, aliada à possibilidade de simulação das máquinas associativas em ambientes convencionais, justifica em parte, a baixa prioridade dada atualmente ao desenvolvimento de computadores associativos comerciais.

II.7. "ARRAY PROCESSORS"

As aplicações que envolvem processamento científico, exigem alguns requisitos especiais de um sistema computacional. Alta performance em cálculos repetitivos, manipulação de grandes conjuntos de dados, boa precisão em cálculos sucessivos, representação de valores em um largo espectro numérico e um razoável conforto para programação são exemplos de características importantes. De uma forma geral, os algoritmos para cálculo científico consistem de longas sequências de somas, multiplicações, multiplicações-somas, etc., envolvendo números reais e complexos. A viabilização destas aplicações está veiculada a um tratamento eficiente destas sequências, principalmente nos casos de processamento em tempo real.

A utilização de computadores de uso geral em aplicações científicas de grande porte, tem normalmente conduzido a uma alta relação custo/performance no resultado final. Se por um lado os requisitos de velocidade são satisfeitos com o emprego de supercomputadores, os custos envolvidos podem tornar esta solução proibitiva. Além disso, outros aspectos como o tamanho físico e a potência consumida precisam ser levados também em consideração. Por outro lado, os sistemas de menor porte, apesar do seu baixo custo, não apresentam um rendimento satisfatório para uma grande quantidade de aplicações científicas.

As restrições citadas, envolvendo computadores de uso geral,

foram responsáveis pelo aparecimento de máquinas mais velozes, destinadas a executar operações científicas específicas. Os "array processors" representam uma parcela significativa deste conjunto. Este termo tem sido ambiguamente citado na literatura (43) para referenciar dois tipos de arquitetura que serão tratadas separadamente neste trabalho.

O primeiro tipo, aqui denominado processador em matriz diz respeito a um computador composto por vários elementos processadores, que podem estar eventualmente dispostos segundo uma topologia matricial. Estes elementos normalmente tem capacidade para realizar operações simultâneas nos dados que manipula, oferecendo soluções bastante poderosas, ainda que a um custo relativamente alto. Os processadores em matriz se enquadram rigorosamente na categoria de máquinas SIMD, já que as instruções afetam simultaneamente todos os elementos processadores de uma mesma maneira. O sistema ILLIAC IV, por exemplo, é uma das mais famosas máquinas deste tipo, tendo sido concebida pela Universidade de Illinois para suportar 256 elementos processadores, divididos em quatro quadrantes. Apesar das operações multi-quadrantes não terem sido implementadas, foi construída uma versão com 64 elementos processadores, controlados por uma unidade central que dispunha de memórias independentes para instruções e dados. O objetivo inicial deste projeto estava voltado para processamento de matrizes, mas havia a possibilidade de se programar o ILLIAC IV para funcionar como uma máquina associativa de 64 elementos.

Um segundo tipo de arquitetura, emprega o termo "array processor" para indicar os tipos de dados que vão ser manipulados. Estes equipamentos (44), doravante batizados de processadores periféricos de matrizes, costumam operar em conjunto com um sistema principal (normalmente um minicomputador) realizando de maneira eficiente operações em dados do tipo matriz. Um exemplo de processador periférico de matrizes bastante conhecido comercialmente é o AP-120B, fabricado pela Floating Point Systems (45).

Os processadores periféricos representam atualmente uma

ferramenta bastante eficiente para acelerar o processamento de aplicações científicas "compute-bound". O baixo custo e o alto grau de maturação alcançado pelo hardware e software dos equipamentos existentes refletem uma experiência acumulada pelos fabricantes e usuários deste tipo de máquina, desde a época que sucedeu a Segunda Guerra Mundial. Uma amostra do potencial deste tipo de ferramenta pode ser observada na Tabela II.3, onde o AP-120B se destaca por seu baixo custo relativo.

O projeto de um processador periférico de matrizes é orientado para o acoplamento a um computador hospedeiro, com a intenção de aumentar a performance do mesmo em tarefas computacionais específicas. A composição interna de um equipamento destes, normalmente inclui várias unidades aritméticas independentes, empregadas na realização de somas, multiplicações, etc.. Uma alta performance é usualmente alcançada através da exploração de paralelismo e/ou "pipelining" na estrutura.

Nos processadores periféricos de matrizes, além da separação dos meios de armazenamento de instruções e dados, verifica-se a existência de memórias para cada tipo de dado. As palavras de instrução, que são praticamente um microcódigo da máquina, são normalmente mais largas que as palavras de dados. As diversas combinações de operações, visando uma máxima utilização das diversas unidades aritméticas tornam vantajosa a criação de memórias de dados independentes. Resumindo, a diferença de tamanho das palavras, combinada com a necessidade de haver paralelismo no acesso às informações, conduz normalmente à separação das memórias internas dos processadores periféricos de matrizes, de acordo com a sua aplicação.

O desenvolvimento de sistemas digitais de computação baseados em processamento periférico é bastante explorado por duas grandes linhas de investigação (46): o processamento matemático de vetores, voltado principalmente para a resolução de equações diferenciais parciais e o processamento digital de sinais (47), que manipula com dados serializados no tempo.

A primeira linha de pesquisa é fundamental na análise de

MÁQUINA	VELOCIDADE MÁXIMA TEÓRICA (M FLOPS)	CUSTO DA INSTALAÇÃO (US $ x 10^6)	CUSTO RELATIVO ($ / FLOP)
CRAY - 1	140	8	4.6
STAR - 100	50	8	12.8
ILLIAC IV	80	10	10.0
CDC 7600	15	3	16.0
AP - 120 B	12	0.15	1.0
IBM 370-168	4	2	40.0
CDC 6600	3	1	26.7
VAX - 11/780	0.5	0.20	32.0
PDP- 11/70	0.2	0.15	60.0

TABELA II.3 - TABELA DE COMPARAÇÃO ENTRE COMPUTADORES DE USO GERAL E "ARRAY PROCESSORS".

sistemas físicos associados a diversas áreas como meteorologia, hidrodinâmica, fissão nuclear, transferência de massa e calor, poluição do ar e da água, etc.. A utilização de computadores digitais para solução destes problemas requer uma discretização dos pontos continuamente distribuídos no domínio espaço-tempo. Quanto maior o número de pontos levados em consideração, menores são as possibilidades de erros na solução final.

As séries temporais de dados, resultantes da digitalização de informações analógicas criaram uma nova classe de problemas, que estimularam o desenvolvimento de processadores periféricos de matrizes. Em aplicações tais como radar, sonar, prospecção geofísica de petróleo, processamento de imagem, etc., são aquisitados e tratados valores analógicos de tensão, corrente, intensidade de luz, deslocamento, etc.. Estas variáveis são amostradas sincronamente e convertidas para um formato digital adequado ao seu processamento e armazenamento. O processamento - que pode englobar filtragem, correlação, análise espectral, etc. - deve em grande parte dos casos ser executado em tempo real. Consequentemente, o desempenho computacional requerido excede a capacidade dos computadores sequenciais mais comuns. Tipicamente, os processadores periféricos de matrizes são bastante eficientes na realização das operações normalmente utilizadas em processamento de sinais.

As ferramentas de programação empregadas nos processadores periféricos de matrizes são, na maioria, de baixo nível, sendo bastante comum a utilização de microprogramação. A razão disso é a necessidade de alcançar a máxima eficiência no processamento predominantemente paralelo. Ainda não há um meio de representação simbólica, quer seja com linguagens de alto nível ou código "assembly"-equivalente, que permita um controle eficiente e preciso das potencialidades suportadas pelas arquiteturas. Sendo assim, a (micro) programação destas máquinas torna-se tarefa bastante árdua, já que o usuário precisa entender os mecanismos de transferência simultânea de dados através dos caminhos paralelos. Além disso, é preciso que sejam considerados na execução de cada operação os retardos relativos introduzidos por cada módulo. Em virtude das limitações naturais

já citadas (ver Seção II.5), a eficiência alcançada pelos processadores periféricos de matrizes é baixa para os casos não triviais, sendo considerado excelente valores acima de cinquenta por cento da velocidade máxima teórica.

Os usuários podem escapar do "fantasma" da microprogramação empregando bibliotecas matemáticas fornecidas pelos fabricantes. Estas unidades de microcódigo, compostas por até algumas centenas de instruções otimizadas, podem ser introduzidas no programa de aplicação por meio de chamadas de alto nível.

III. ARQUITETURAS MIMD UTILIZANDO MICROPROCESSADORES

As arquiteturas MIMD são caracterizadas pela presença de duas ou mais unidades processadoras destinadas a realizar operações múltiplas em conjuntos distintos de dados. Embora sob o ponto de vista técnico não exista nenhuma restrição quanto ao porte destas unidades, será dada maior ênfase às estruturas MIMD compostas por microprocessadores. Esta escolha reflete a tendência atual em agregar componentes VLSI com o objetivo de obter arquiteturas mais poderosas. Esta perspectiva vem se tornando mais promissora após o lançamento de microprocessadores com performance comparável à máquinas de médio porte (48).

Embora os componentes VLSI estejam ocupando uma ampla faixa no espectro de aplicações de sistemas digitais, resta saber se estes componentes poderão ser agrupados para construir grandes sistemas genéricos de computação. Em outras palavras, caberia aqui uma pergunta: pode um determinado conjunto de microprocessadores ser interconectado para executar tarefas que necessitam de uma velocidade global de ordem de vários milhões de instruções por segundo? Atualmente não há uma resposta definitiva para esta pergunta, embora existam diversas razões para acreditar que grandes sistemas a multimicroprocessadores sejam viáveis a médio prazo. Estes argumentos já foram bastante discutidos na literatura em geral, e estão ligados à capacidade de realização de sistemas com boas características de performance/custo, disponibilidade, modularidade, flexibilidade, etc..

Uma diferença em performance de aproximadamente duas ordens de grandeza separa atualmente os microprocessadores das máquinas de grande porte empregadas em processamento numérico pesado. A seguir são apresentados os principais problemas que os projetistas estão enfrentando para tentar reduzir esta diferença (49).

- Decomposição de tarefas: Como dividir as tarefas entre os processadores? A decomposição deve ser automática (através de

compiladores ou rotinas chamadas em tempo de execução) ou deve ser deixada para o programador?

- Interferência: Após a decomposição das tarefas entre os diversos processadores, quais as técnicas a serem empregadas para minimizar a contenção pelos recursos de memória e E/S?

- Estruturas de interconexão: Quais são os tipos mais eficientes de conexão entre processador/memória e processador/processador? Quais os protocolos que devem ser associados a cada caso?

- Estrutura de software: Que estruturas são mais adequadas para sistemas com dezenas ou centenas de processadores? Problemas importantes nesta área são gerenciamento de recursos, distribuição de software, proteção e confiabilidade.

- Mecanismo de mapeamento de endereços: Que mecanismos são apropriados para efetuar a conversão de endereços virtuais para reais? Estes mecanismos devem permitir o compartilhamento de código e dado ao mesmo tempo em que devem garantir níveis adequados de proteção e eficiência.

- Prevenção de retardos infinitos ("deadlocks"): A disputa pelos recursos dá margem a uma situação perigosa em que todos os processadores ou uma parte deles estejam impedidos de executar, por estarem esperando a liberação de um recurso que não irá ocorrer. Esta situação deve ser evitada.

- Tolerância a falhas: Quais as estruturas de hardware e software que tirarão o máximo proveito da capacidade de recuperação inerente às estruturas com múltiplos processadores?

- Entrada e saída: Como os dispositivos de entrada e saída, especialmente memórias de massa, devem ser integrados a um sistema a multi-microprocessadores?

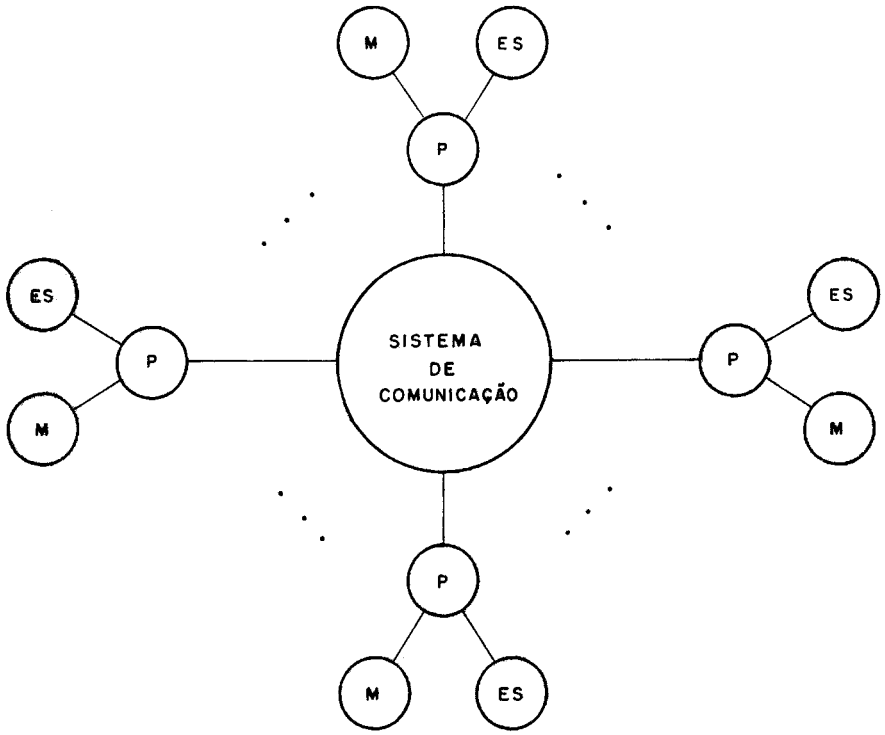
A investigação destas e de outras questões (50) deve ser

analisada segundo parâmetros atualizados, já que diversos fatores, como o custo de componentes por exemplo, tem sofrido constante modificações através do tempo. As duas arquiteturas mostradas na Figura III.1 diferem quanto ao grau de acoplamento existente entre os elementos que a compõe. Esta diferença reflete na verdade dois conjuntos de problemas e soluções que estão sendo investigados para atender a dois tipos diferentes de aplicação: as que exigem distribuição da inteligência para execução de tarefas independentes e as que buscam na aproximação dos componentes uma maior eficiência na resolução de problemas mais interligados. A seguir serão analisadas mais detalhadamente estas duas alternativas.

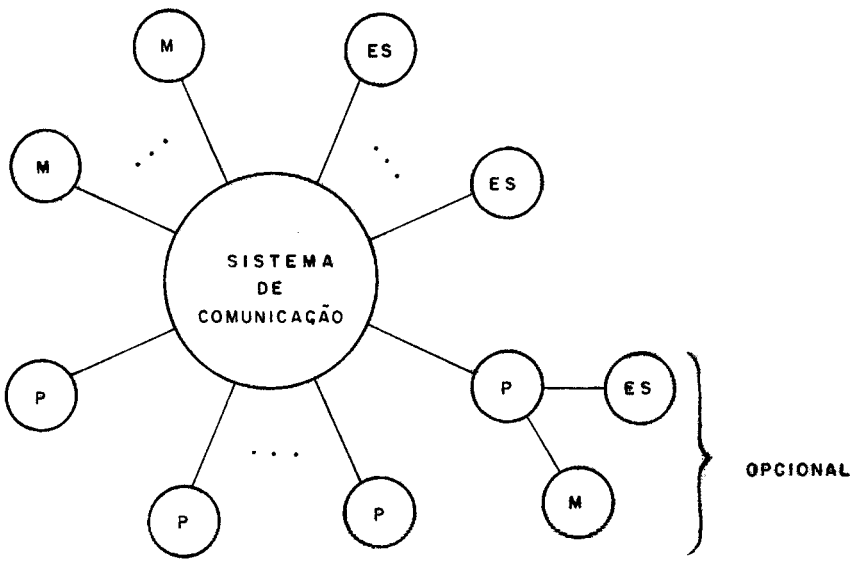
III.1. MULTIPROCESSADORES

As pesquisas iniciais na área de multiprocessamento foram direcionadas para aplicações militares e de controle. O objetivo inicial era explorar a grande disponibilidade que poderia ser obtida com a interligação de um grupo de unidades idênticas, cada uma capaz de executar as mesmas tarefas. Somente mais tarde é que os projetistas começaram a capitalizar os ganhos em performance que poderiam ser alcançados por tais arquiteturas. As maiores dificuldades com que os pesquisadores do assunto se depararam estavam relacionadas com o software destas máquinas. Em alguns casos, os computadores eram comercializados antes mesmo que houvesse disponibilidade de um sistema operacional adequado.

Um computador baseado em multiprocessamento, é tido como uma máquina formada por dois ou mais processadores com capacidades aproximadamente semelhantes, agrupadas de forma tal que todos possam compartilhar a memória e os periféricos principais do sistema. Os processadores são controlados por um sistema operacional que deve prover uma estreita interação entre eles, tanto nos diversos níveis de software quanto nas funções de hardware (51).



a. - ESTRUTURA FRACAMENTE CONECTADA: O PRINCIPAL RECURSO COMUM É O PRÓPRIO SISTEMA DE COMUNICAÇÃO.



b. - ESTRUTURA FORTEMENTE CONECTADA: OS RECURSOS PRINCIPAIS DO SISTEMA SÃO LIGADOS DIRETAMENTE AO SISTEMA DE COMUNICAÇÃO.

FIGURA III.1 - FORMAS DE INTERLIGAÇÃO DE PROCESSADORES, MEMÓRIAS E E/S.

III.1.1 REQUISITOS BÁSICOS

Da definição pode-se concluir, como primeiro requisito para um sistema baseado em multiprocessamento, que o fluxo de dados deve ser organizado de uma forma tal que permita que um processador possa acessar uma informação em qualquer parte do sistema (memória ou e/s). A única ressalva aceitável e vantajosa em determinadas situações ocorre nos casos em que recursos locais são reservados para um determinado processador. Dependendo da eficiência e segurança desejadas na arquitetura final, outros requisitos se fazem necessários:

- Mútua exclusão: Para garantir a integridade dos dados deve ser previsto no hardware um acesso exclusivo aos recursos compartilhados.

- Endereçamento virtual: Um esquema de endereçamento que utilize referências lógicas em vez de físicas permite maior flexibilidade no gerenciamento da alocação de recursos.

- Deteção e recuperação de falhas: Um processador deve ser capaz de sinalizar ou interromper qualquer outro. Isto permite, entre outras coisas, a verificação e a manutenção do bom estado de funcionamento do sistema. Em caso de pane, as informações que estavam sendo acessadas em um processador devem poder ser transferidas para o elemento que irá continuar a execução. A recuperação exige ainda que um processador possa reinicializar outro que esteja potencialmente operacional.

III.1.2. ESTRUTURAS DE INTERCONEXÃO

Os multiprocessadores podem ser classificados de acordo com a sua estrutura de interconexão. Apesar da grande variedade de organizações existentes, é possível identificar três topologias fundamentalmente distintas, que podem ser consideradas como uma base ortogonal razoavelmente representativa do espaço que engloba a maioria das arquiteturas de multiprocessamento. São elas:

- A barra comum multiplexada no tempo;
- O sistema matricial de chaves em barras cruzadas e
- As memórias multi-portas.

A ligação através da barra comum, ilustrada na Figura III.2 constitui um dos meios mais simples e baratos de interconexão. O barramento é formado por grupos de condutores que contém informações de endereço, dado, sincronismo e controle. O acesso aos recursos comuns pode ser coordenado através de um controlador dedicado ou então por meio de uma política de distribuição de prioridade aos consumidores. O primeiro caso é caracterizado pela atuação centralizadora de um árbitro único, enquanto que no segundo o gerenciamento da multiplexação é incorporado ao próprio barramento. Na maioria dos casos, em qualquer um dos dois métodos de acesso, a comunicação é realizada entre dois elementos de cada vez num esquema mestre - escravo.

Alguns exemplos de barramentos são o Multibus da INTEL (IEEE-P796) que é bastante utilizado em microcomputadores, o GPIB (IEEE-488) que vem sendo aplicado à instrumentação e o Unibus da Digital Equip. Co. que embora não seja utilizado em multiprocessamento apresenta características adequadas para tal atividade.

As limitações registradas nas taxas de transferência de dados representam uma grande desvantagem dos barramentos. A causa principal desta limitação é a multiplexação do sistema de interconexão entre os diversos consumidores de recursos. Este fator restringe em algumas unidades apenas a quantidade de processadores viáveis a um sistema interligado por barramento.

Algumas modificações vem sendo introduzidas para aliviar a contenção no barramento comum. Uma delas consiste em permitir a comunicação entre diversos elementos de uma só vez. Consegue-se assim multiplicar a taxa de transferência por um fator igual à quantidade de componentes que estão "escutando" um determinado comando ou mensagem. Uma segunda técnica prevê a introdução de

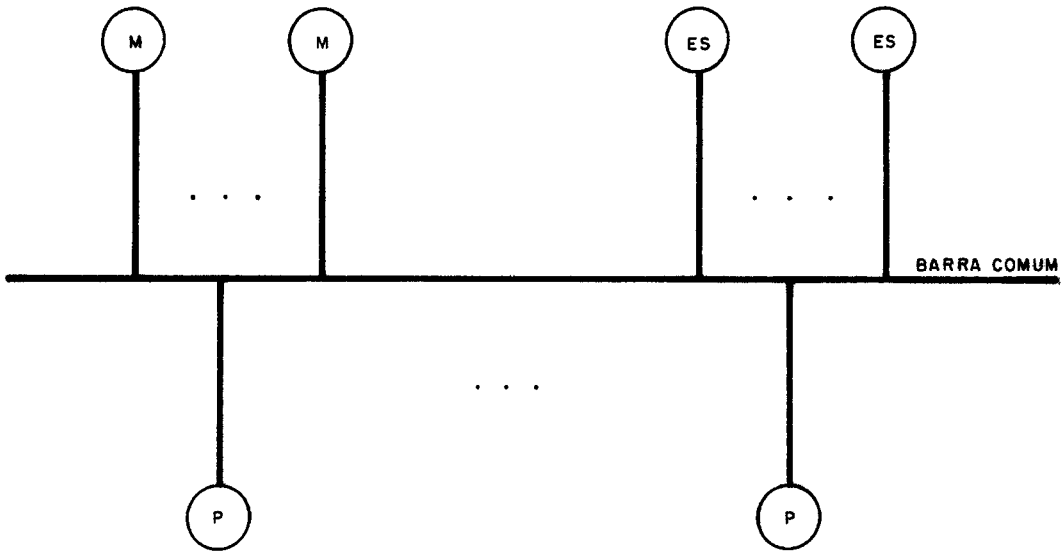


FIGURA III. 2 - ESTRUTURA EM BARRA COMUM. OS RECURSOS SÃO MULTIPLEXADOS NO TEMPO.

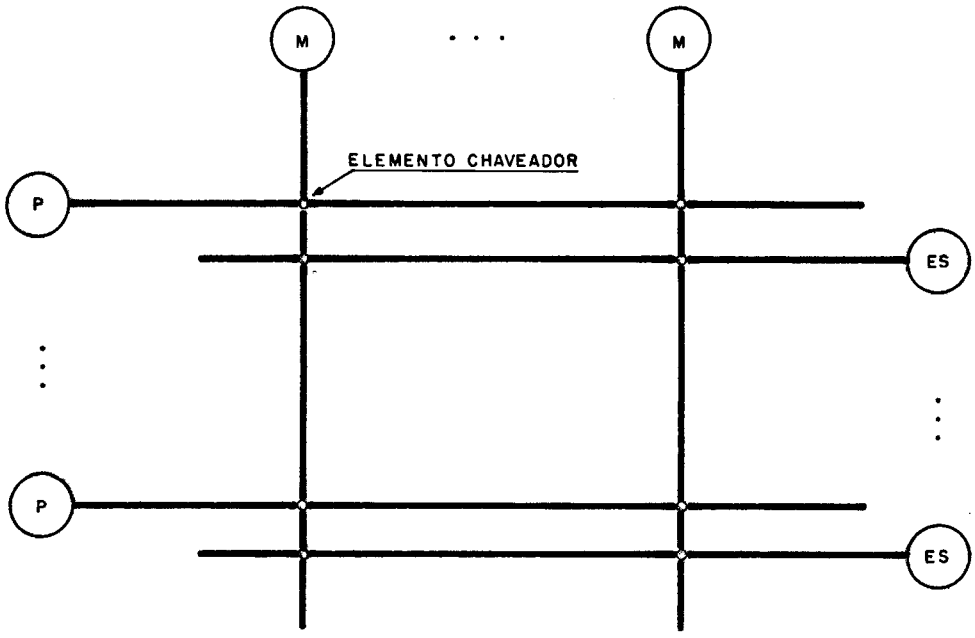


FIGURA III. 3 - SISTEMA MATRICIAL DE CHAVES EM BARRAS CRUZADAS.

barramentos alternativos. Em face às dificuldades de implementação de múltiplos caminhos globais de uso geral, é usual que sejam criados barramentos dedicados. Algumas implementações canalizam separadamente as informações para processadores, memórias e periféricos, enquanto que outras criam vários níveis hierárquicos para o tráfego de dados.

Um segundo sistema de interconexão é formado por barramentos cruzados, que podem ser logicamente conectados para interligar os processadores e periféricos existentes na estrutura, conforme mostra a Figura III.3 num exemplo típico.

Toda a potencialidade desta estrutura é fruto da alta capacidade de transferência que pode ser obtida através da configuração de várias rotas independentes entre os diversos grupos de elementos que precisam se comunicar.

O não bloqueamento decorrente das transferências simultâneas suportadas pelo sistema matricial de barras cruzadas não impede que haja contenção na alocação de recursos. Dois ou mais consumidores continuam podendo tentar o acesso a um recurso comum, como no caso do barramento. Nesse caso, é necessária a interferência de um arbitrador para resolver a disputa. Normalmente esta lógica é acoplada ao controle de comutação das chaves da malha.

Os barramentos redundantes garantem normalmente boas taxas de confiabilidade aos sistemas interligados por barras cruzadas comutáveis. Entretanto, esta e as outras vantagens já citadas são parcialmente obscurecidas pela grande complexidade do sistema de chaveamento. O número de elementos comutadores é proporcional ao produto entre a quantidade de processadores e unidades de memória, o que torna proibitiva a extensão deste tipo de interconexão a sistemas de grande porte.

A terceira classe de topologias de interconexão para multiprocessamento é o sistema de memórias multi-portas. Estas memórias contém mais de um acesso, que são controlados por uma lógica que resolve os problemas de arbitragem. As memórias

multi-portas também podem ser usadas em processadores simples, visando uma maior eficiência na realização de operações de E/S.

No caso mais geral, cada memória dispõe de uma porta para cada processador ou controlador de e/s, como exemplificado na Figura III.4. A grande quantidade de fiação daí decorrente, e a baixa capacidade de expansão limitam o número de processadores normalmente utilizados neste tipo de estrutura.

Os esquemas de multiprocessamento que empregam memórias multi-portas apresentam muito bom desempenho. Entretanto, o custo das unidades de memória é maior que nos outros dois casos, principalmente se muitos caminhos de acesso precisam ser suportados.

Cabe aqui alguns comentários sobre outras estruturas de interligação adequadas a multiprocessamento. Como pode ser observado, as três topologias citadas apresentam características bastante diferentes se considerarmos parâmetros como custo, velocidade, confiabilidade, expandibilidade e facilidade de programação. Atualmente um volume considerável de pesquisas está voltado para o desenvolvimento de outras estruturas que permitam a cooperação eficiente de um grande número de processadores (52).

III.2. REDES LOCAIS DE COMUNICAÇÃO

A análise das redes aqui consideradas irá se deter mais nas estruturas para interconexão local, onde a concentração geográfica acarreta um baixo custo de construção dos meios de transmissão e altas taxas de comunicação são alcançadas.

As redes locais devem ser diferenciadas das redes de comunicação à longa distância (como o ARPANET), embora existam problemas comuns que possam ser resolvidos em conjunto. Projetadas para realizar a interligação de pontos situados à distâncias superiores a um quilômetro, as redes de longo alcance tem características técnicas e econômicas bastante diferentes das

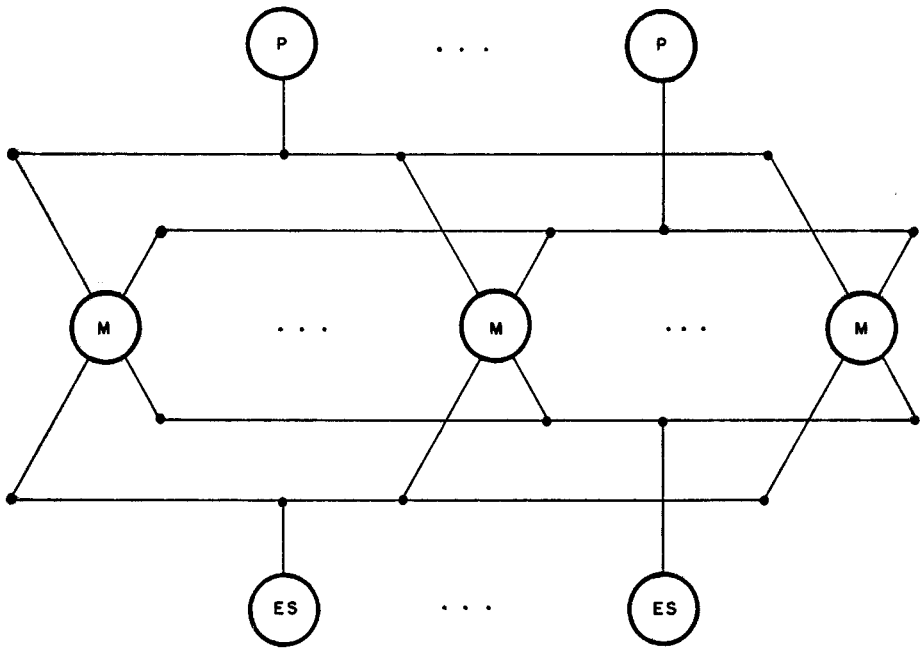


FIGURA III. 4 - ARQUITETURA BASEADA EM MEMÓRIAS MULTI-PORTAS.

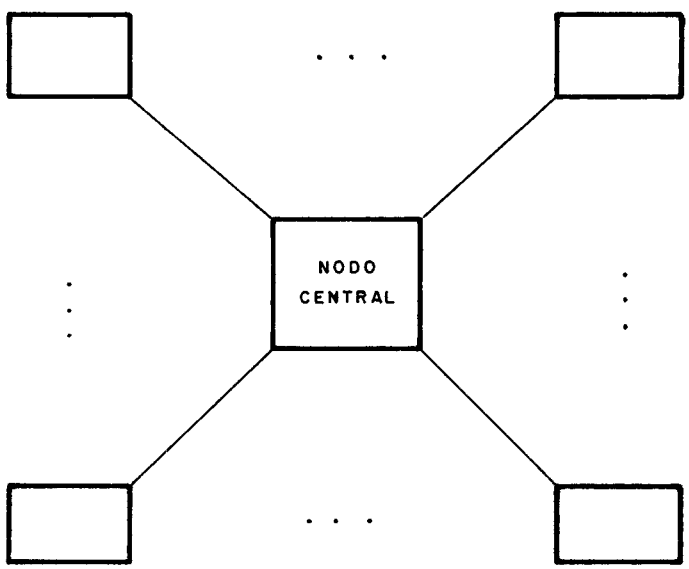


FIGURA III. 5 - REDE CONFIGURADA EM ESTRELA. O ELEMENTO CENTRAL TOMA A INICIATIVA DOS DIÁLOGOS.

redes locais.

Outros tipos de meios de comunicação que poderiam ser confundidos com as redes locais são as estruturas para multiprocessamento citadas no item anterior. As redes locais não podem, por exemplo, ser tratadas como um longo barramento (53), porque além das diferenças topológicas, tecnológicas e geográficas, há que se levar em consideração alguns aspectos filosóficos fundamentais: o barramento de um computador foi concebido para conectar componentes imprescindíveis ao funcionamento do mesmo, ou seja, é difícil imaginar um computador operando sem o seu barramento. Uma rede, contudo, tem a função de interligar diversos módulos autônomos, capazes de operar de forma quase independente.

Os componentes que compõem uma rede local são basicamente:

- o meio físico de comunicação, responsável pela propagação das informações entre os membros da rede;
- o mecanismo de controle, que age sobre o meio de comunicação;
- o mecanismo de adaptação da rede ao computador e
- um conjunto de protocolos que, em diversos níveis, regem a transferência das informações através dos componentes da rede.

A maioria das redes locais existentes utiliza-se de cabos coaxiais ou fios trançados como meio de comunicação. Outros meios estão sendo investigados, como por exemplo, os cabos de redes privadas de televisão (CATV) e as fibras-óticas. Estas últimas apresentam algumas características bastante desejáveis em uma rede, quais sejam: viabilização de altas taxas de comunicação (na faixa de 1 a 20 Mbit/s) em distâncias de alguns quilômetros, alta imunidade a ruído e isolamento elétrico. Algumas topologias, entretanto, ainda não podem ser implementadas facilmente com esta tecnologia, em virtude de problemas ainda não resolvidos, como o T-ótico.

A atuação dos mecanismos de controle sobre os meios de comunicação definem a habilidade com que são transferidas as informações de um ponto ao outro de uma rede. As topologias mais

importantes a serem consideradas, são aquelas que apresentam processos simples de direcionamento de mensagens na rede. São elas:

- a estrela mostrada na Figura III.5, que é indicada para os casos em que todas as comunicações incluem um elemento central, como no caso de uma rede de terminais em um sistema "time-sharing";

- o anel, apresentado na Figura III.6, em que as mensagens são passadas de nodo em nodo, segundo ligações unidirecionais, até alcançar o nodo-destino e

- o barramento, que faz com que as mensagens sejam veiculadas em todas as direções na rede, até que o nodo-destino possa recebe-la (Figura III.7).

Os mecanismos que controlam as redes em anel e os barramentos precisam determinar qual nodo tem permissão para transmitir informações num determinado instante. No caso da estrutura em estrela, basta que o elemento central se encarregue desta tarefa, dando chance a um nodo secundário de cada vez.

Várias estratégias de controle adequadas às estruturas em anel são disponíveis atualmente. Todas partem do princípio que a permissão para o uso da rede deve ser passada de nodo em nodo. Tanto podem ser usados recursos de hardware (como no "daisy chain"), quanto de software (como nos casos do "control token" e do "message slot") para definir as prioridades de acesso ao anel.

Os barramentos também podem ser administrados de diversas formas. Existem, por exemplo, métodos de acesso randômico que resolvem os conflitos causados por colisão de mensagens somente no instante em que eles ocorrem (54, 55).

As três topologias mostradas podem ainda combinar as diferentes estratégias de controle apresentadas, possibilitando que uma estrutura do tipo barramento possa, por exemplo, ser controlada

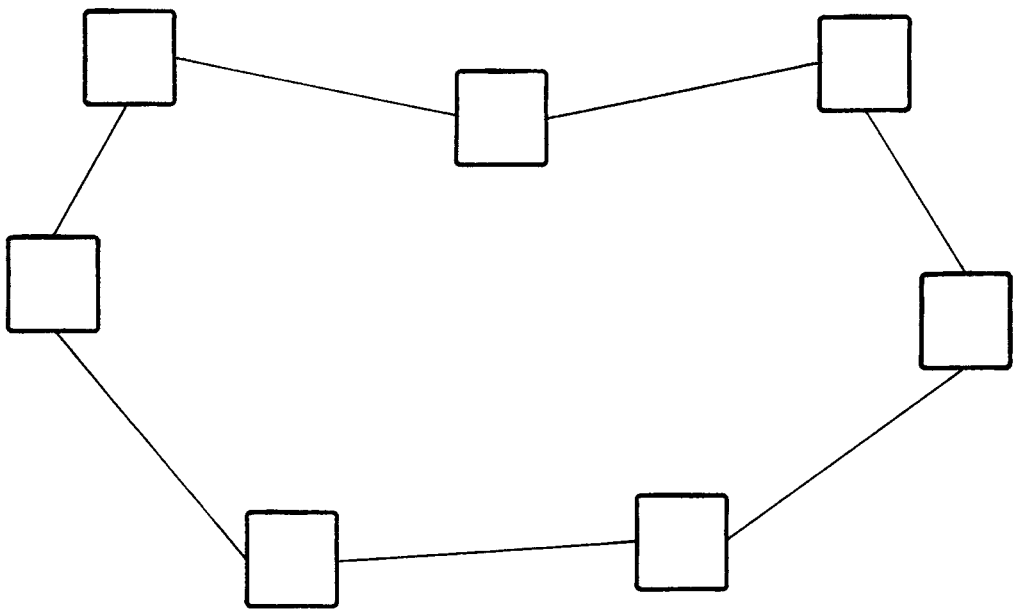


FIGURA III. 6 - REDE CONFIGURADA EM ANEL. A INFORMAÇÃO CIRCULA DE UM NODO PARA OUTRO.

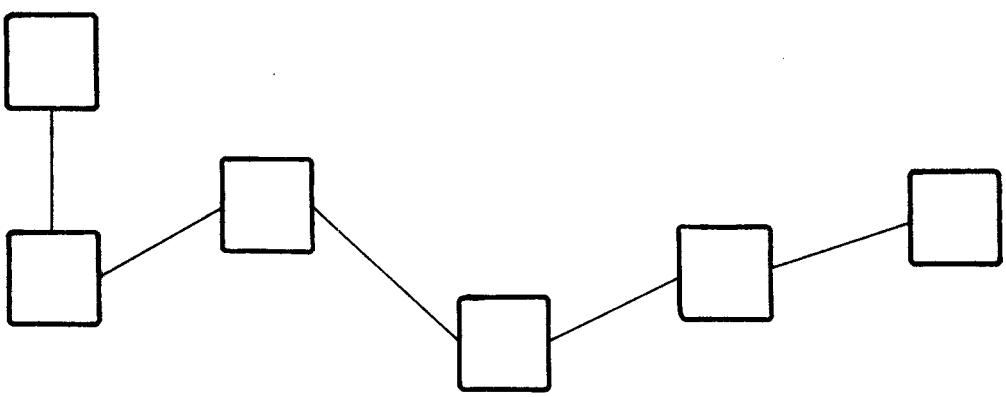


FIGURA III. 7 - REDE CONFIGURADA EM BARRAMENTO. A INFORMAÇÃO É VEICULADA EM TODAS AS DIREÇÕES.

por uma estratégia em anel (como o "control token") (56). Outras possibilidades são citadas em Clark et al (53).

A principal motivação para o uso de topologias em anel e barramento, está relacionada aos problemas de confiabilidade inerentes à estrutura em estrela. Tanto a topologia quanto a estratégia de controle desta última rede dependem demasiadamente do bom funcionamento dos componentes associados ao nodo central. Ainda assim, a estratégia de controle para as estruturas mais distribuídas precisam considerar a ocorrência de falhas nos nodos que detem o controle da rede. Sob este aspecto, a estratégia de acessos randômicos leva vantagem sobre as estratégias em anel citadas, já que é preciso prever a possibilidade de destruição da entidade responsável pela rede. Por exemplo, um "token" de controle de uma rede em anel pode ser destruído por um ruído transiente no meio da transmissão, fazendo com que sejam interrompidas as comunicações em todos os nodos da estrutura. Nas redes com acesso do tipo CSMA/CD ("Carrier Sense Multiple-Acess with Collision Detection"), uma falha tem o mesmo efeito de uma colisão, e é tratada automaticamente como tal.

O hardware de uma rede local é caracterizado por uma alta relação performance-custo. A adaptação de um processador à uma rede exige normalmente, dos circuitos de interligação, funções de reconhecimento de endereços e de controle das transmissões. A crescente utilização de processadores baratos e a alta relação performance-custo exigida das redes locais tem motivado o desenvolvimento de unidades de adaptação bastante eficientes. As tecnologias de integração de circuitos em larga escala já permitem a implementação de circuitos com estas características.

A transferência de informações através de redes obedecem a um conjunto de regras, destinadas a organizar as transações efetuadas. Estes conjuntos de regras, que recebem o nome de protocolos, são responsáveis pelo funcionamento da rede nos seus diversos níveis de operação. Exemplos de atribuição dos protocolos são:

- o estabelecimento das convenções necessárias ao entendimento das informações trocadas. Incluem-se nesta lista, por exemplo, os níveis de tensão e faixas de frequência dos sinais que transitarão na rede;

- a identificação dos elementos que compõem a rede, para que os nodos possam se endereçar mutuamente;

- o estabelecimento de uma política destinada a controlar o fluxo de informações nas diversas seções da rede;

- a deteção e recuperação de erros causados pela rede;

- a sequenciação das informações no nodo receptor na ordem em que elas foram transmitidas e

- a criação de regras para estabelecimento e término de conexão entre usuários. O conceito de conexão permite que sejam otimizados os recursos da rede de um modo geral.

Desde o nível dos meios físicos de comunicação até as camadas mais externas de uma rede, existem vários serviços a serem executados. Estes serviços são normalmente organizados de uma forma hierárquica, de forma que um determinado nível possa usufruir dos serviços já suportados pelas camadas mais inferiores da rede. As camadas variam desde o nível de enlace físico que envolve circuitos de hardware, até os protocolos de alto nível que oferecem aos usuários de uma rede serviços mais próximos de suas reais necessidades (57).

IV. SISTEMA-P UCSD

O Sistema-p UCSD foi projetado pelo "Institute for Information System" da Universidade da Califórnia em San Diego - USA. O objetivo inicial dos alunos de graduação e pós-graduação liderados por Kenneth Bowles, era substituir, nos cursos iniciais de computação, o computador de grande porte da universidade por diversos microcomputadores. Além de permitir um grau mais estreito de interação com os usuários iniciantes, esta idéia iria resolver alguns problemas administrativos do Centro de Computação, em virtude da decorrente descentralização dos serviços.

Num curto espaço de tempo, o sistema passou a ser utilizado para o desenvolvimento de programas de uma maneira geral, sendo empregado inclusive na geração das versões subsequentes do mesmo sistema. O pequeno grupo da universidade não esperava que o sistema fosse despertar algum interesse comercial. Contudo, a procura ao "pacote" foi tão grande que o "projeto Pascal" acabou sendo entregue a uma firma particular para comercialização: a Softech Microsystems. Esta firma, que recebeu da universidade uma versão preliminar, vem introduzindo diversos melhoramentos no Sistema-p, visando um alargamento do seu espectro de aplicações. Atualmente a versão IV.0 inclui algumas extensões, como programação concorrente, gerenciamento mais otimizado da memória, novas linguagens, etc.

O hardware típico para que seja alcançado um funcionamento eficiente do Sistema-p inclui um microcomputador com pelo menos 48K bytes de memória, um terminal e um dispositivo de memória de massa. Este último é normalmente composto por uma ou mais unidades de disco flexível, existindo mais recentemente uma tendência para utilização da tecnologia Winchester de discos rígidos. Outros recursos suportados incluem impressora, terminal gráfico, plotter e uma rede de comunicação entre computadores.

O software disponível no Sistema-p inclui compiladores, montadores, editores de texto, ferramentas para manipulação de

arquivos em disco e link-edição de programas, além de utilitários de uso geral. Uma característica bastante importante do software desenvolvido pela UCSD é a sua portabilidade, pois todo o sistema está baseado em um pseudo-processador que define a chamada máquina-p. As linguagens de alto nível (Pascal, Fortran e Basic entre outras) são compiladas para uma linguagem intermediária (código-p), correspondente ao conjunto de instruções da máquina virtual. Para executar este código em um hardware real é necessário emular a máquina-p. Para tanto, podem ser usados um interpretador que processe as operações em tempo de execução, um gerador de código que realize a tradução antes da execução ou ainda uma máquina especial que execute diretamente o código-p.

Os processadores mais usados para emular a máquina-p atualmente são o Intel 8080 e 8085, o Zilog Z80, o Digital PDP-11 e LSI-11, o Motorola 6800 e 6809, o Texas 9900, o Fairchild 6502 (Apple) e mais recentemente os microprocessadores de 16 bits da Intel (8086 e 8088), Motorola (68000) e Zilog (Z8000). Um processador especial produzido pela Western Digital, implementou em firmware a máquina-p (Pascal Microengine). A diferença existente entre as diversas configurações de hardware citadas está no emulador da máquina-p, pois todo o software escrito em linguagem de alto nível é intercambiável entre as versões.

A interação do usuário com o Sistema-p é orientada por uma árvore de comandos semelhante à da Figura IV.1. Esta árvore estabelece as alternativas disponíveis em cada nível hierárquico, através de uma linha atualizada de comandos, presente na tela do terminal. Os comandos são ativados por uma única tecla, e resultam ou na execução da função selecionada ou na apresentação de uma outra linha de comandos correspondente a um nível inferior da árvore. Maiores informações sobre os comandos disponíveis atualmente no Sistema-p podem ser encontrados no manual de usuários da Softech (58 e 61).

IV.1 ORGANIZAÇÃO DO SISTEMA-P

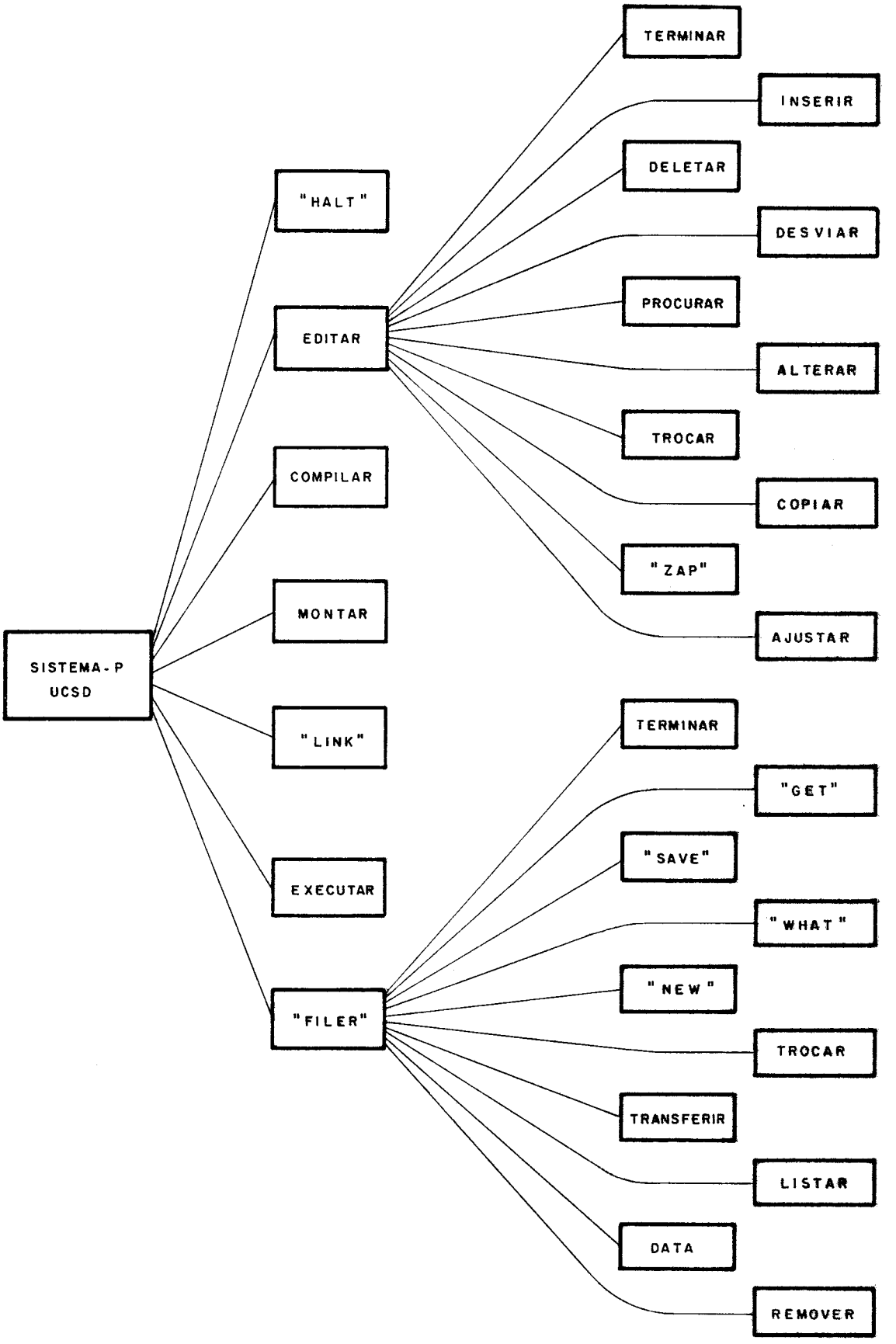


FIGURA IV. I - ÁRVORE DE COMANDOS SIMPLIFICADA.

De acordo com o Capítulo de Introdução, um dos objetivos deste trabalho é adaptar a máquina-p UCSD para uma arquitetura mais apropriada ao processamento paralelo. Para isso será utilizada a versão original do sistema (61) produzida pelo grupo universitário da Universidade da Califórnia. Apesar de já existirem novas versões do Sistema-p, duas razões levaram à escolha da versão I.5 como ponto de partida para o presente trabalho: a disponibilidade dos códigos fontes das principais partes do pacote e descomprometimento com alguma estrutura para processamento concorrente já pré-estabelecida.

A emulação da máquina-p, na versão I.5, é baseada em um microprocessador que realiza a interpretação do código intermediário em tempo de execução. Como etapa inicial do presente trabalho, e visando a utilização de microprocessadores de 16 bits na futura máquina-p estendida, foi contruído um interpretador para o microprocessador Intel 8086. Este interpretador foi baseado nos originais recebidos da UCSD, que incluiam programas equivalentes para os microprocessadores Intel 8080 (ou Zilog Z80) e o DEC LSI-11.

A Figura IV.2 apresenta um diagrama em que figuram os principais elementos que compõe o Sistema-p UCSD na versao I.5. A comunicação entre o software de baixo nível, codificado na linguagem de máquina do processador real, e aquele produzido pela compilação dos trechos escritos em Pascal, é realizada através de uma área chamada SYSCOM ("System Communication Area"). Nela estão incluídas as informações específicas para controle e monitoração do ambiente real de funcionamento da máquina-p. Incluem-se nesta região, por exemplo, os caracteres de controle que irão realizar funções especiais no console do sistema, o estado interno da máquina-p, etc.. As outras variáveis globais do sistema são acessadas exclusivamente pelos procedimentos escritos em linguagem de alto nível.

Um grupo de rotinas, que estão sempre residentes na memória, tem a função de executar os procedimentos intrínsecos que são suportados pelas linguagens de alto nível. Incluem-se por exemplo:

SOFTWARE INDEPENDENTE DA IMPLEMENTAÇÃO

```
program pascalsystem ;
var  syscom : ^ syscomrec ;                    (* apontador para a área comum *)
     .
     .                                         (* outras variáveis globais *)
   procedure exercerr (...);                   (* pacotes de rotinas de run-time *)
     .
     .
     .
   segment procedure initialize;               (* procedures do sistema operacional *)
   segment procedure userprogram (...);
   segment procedure debugger (...);
     .
     .
     .
begin                                          (* programa principal *)
     .
     .
     .
end .
```

SOFTWARE DEPENDENTE DA IMPLEMENTACAO

INTERPRETADOR

SYSCOMREC :
ÁREA COMUM PARA COMUNICAÇÃO COM O RESTANTE DO SISTEMA OPERACIONAL.

HARDWARE

FIGURA IV. 2 - ORGANIZAÇÃO GERAL DO SISTEMA - P UCSD (VERSÃO I.5)

- rotinas usadas em operações de e/s, para realizar a conversão de valores numéricos internos em "strings" de caracteres ASCII;
- procedimentos associados à gerência de arquivos em disco (open, close, etc..) e
- rotinas para tratamento de erros de execução.

O programa pascalsystem expandido para o Sistema-p completo seria representado por mais de 6000 linhas em Pascal, o que resultaria em no mínimo 50k bytes de código compilado. Este volume de software foi considerado bastante grande para uma "máquina pequena" e alguns mecanismos foram criados para permitir a sua partição em segmentos. Desta forma, alguns procedimentos do Sistema-p, que incluem funções de inicialização, interação com o usuário, programas de aplicação, etc., foram agrupados em segment procedures. Estes procedimentos compõe o restante do Sistema-p, juntamente com o corpo do programa principal.

IV.2. MÁQUINA-P UCSD

A Figura IV.3 mostra a configuração típica da memória para o Sistema-p UCSD da versão I.5. O código do interpretador bem como a área de comunicação com o sistema operacional (Syscom), situam-se em um trecho independente da memória (um dos extremos). O "stack" e o "heap" são alocados na memória restante, de maneira que ambos cresçam em sentidos opostos. Estas áreas são o meio principal de armazenamento da máquina-p, sendo o "stack" responsável pela alocação do código e dados estáticos dos programas e o "heap" reservado para as variáveis alocadas dinamicamente.

A distribuição do código e dados estáticos no "stack" é feita segundo uma política de segmentação. Esta política permite que o usuário especifique quais procedimentos residirão permanentemente na memória e quais deverão ser carregados durante a execução. A Figura IV.4 apresenta um exemplo de ativação de procedures, em que são mostradas as duas modalidades de funcionamento do sistema. Enquanto que o código da procedure

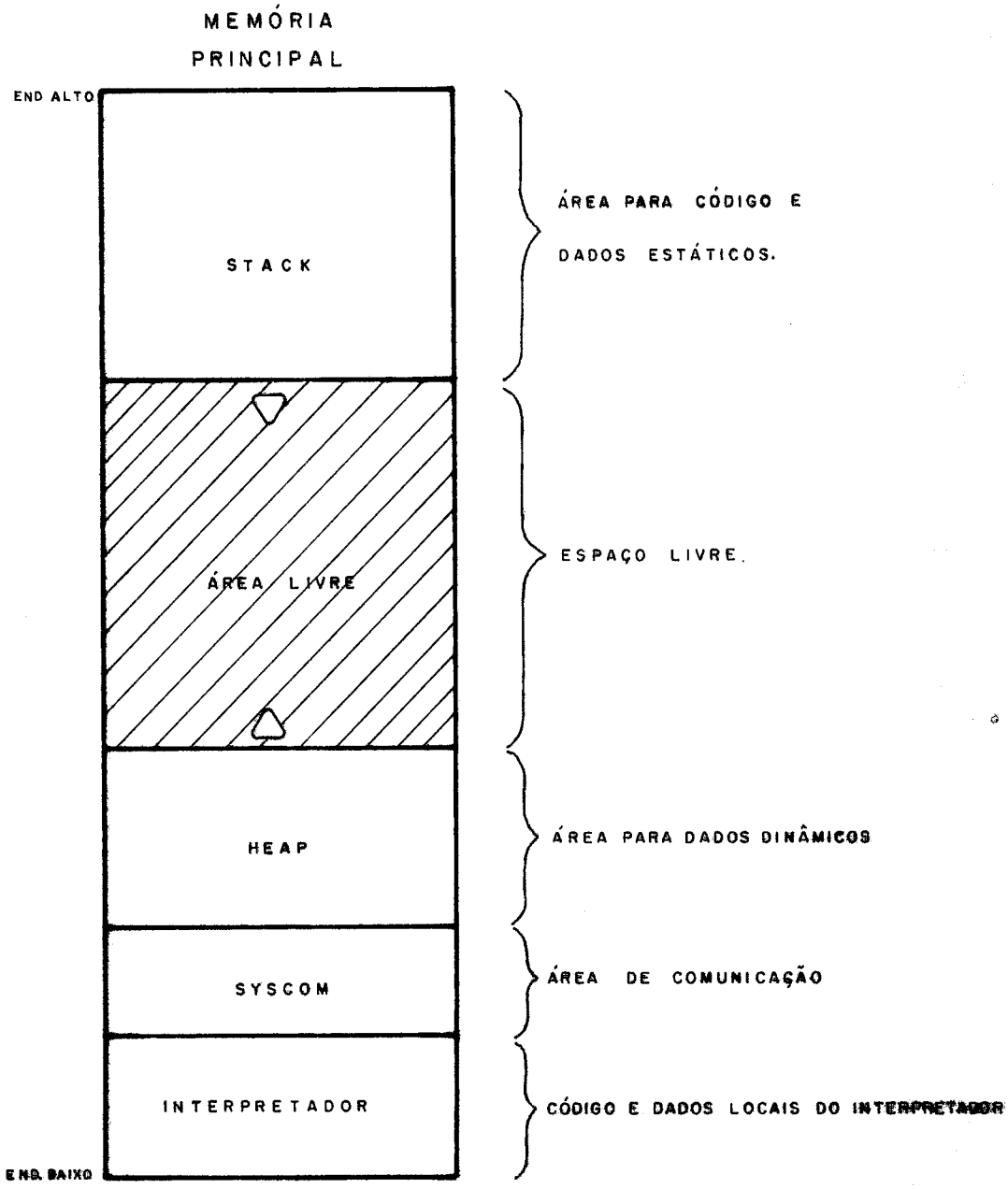


FIGURA IV. 3 - CONFIGURAÇÃO TÍPICA DA MEMÓRIA NO SISTEMA - P (VERSÃO 1.5).

A está sempre na memória, o mesmo não acontece com a segment procedure B.

Quando o segmento B é chamado, o intepretador verifica se ele está presente na memória devido a uma ativação anterior. Se estiver, o controle é passado para ele e a execução continua. Caso contrário, o segmento de código apropriado é carregado do disco, antes da transferência do controle. Quando não houver mais chamadas ativas ao segmento, o seu código é removido da memória. Logicamente, o programa deve ser fragmentado de forma a evitar freguentes chamadas não recursivas a um mesmo segmento, pois senão poderá ocorrer uma grande perda de tempo em operações sucessivas de carregamento de código para a memória. Maiores detalhes sobre os segmentos de código estão disponíveis em manuais da UCSD e da Softech (IV.1 a IV.4).

Conforme mostra a Figura IV.4, o segmento de dados de uma procedure é contruído no topo do stack, no instante em que a mesma é chamada. A parte superior do segmento de dados contém espaço para armazenar as variáveis locais declaradas na procedure. Na parte inferior do segmento de dados fica registrado o estado da máquina-p anterior à chamada da procedure. Este conjunto de palavras de controle é denominado "Mark Stack Control Word" (MSCW) e será detalhado mais adiante.

A máquina-p é caracterizada por um grupo de sete pseudo-variáveis, que atuam como registros internos da máquina virtual. Estes pseudo-registros apontam para posições estratégicas da memória durante a execução de um determinado programa. Considerando ainda o exemplo citado anteriormente, e ampliando o mapa de memória no instante em que a segment procedure B está sendo executada, obtém-se o mapa da Figura IV.5, onde estão representados os pseudo-registros da máquina-p:

SEG - indica qual segmento de código está correntemente ativo;
JTAB - aponta para dentro do segmento de código, indicando o início da procedure que está sendo correntemente executada;

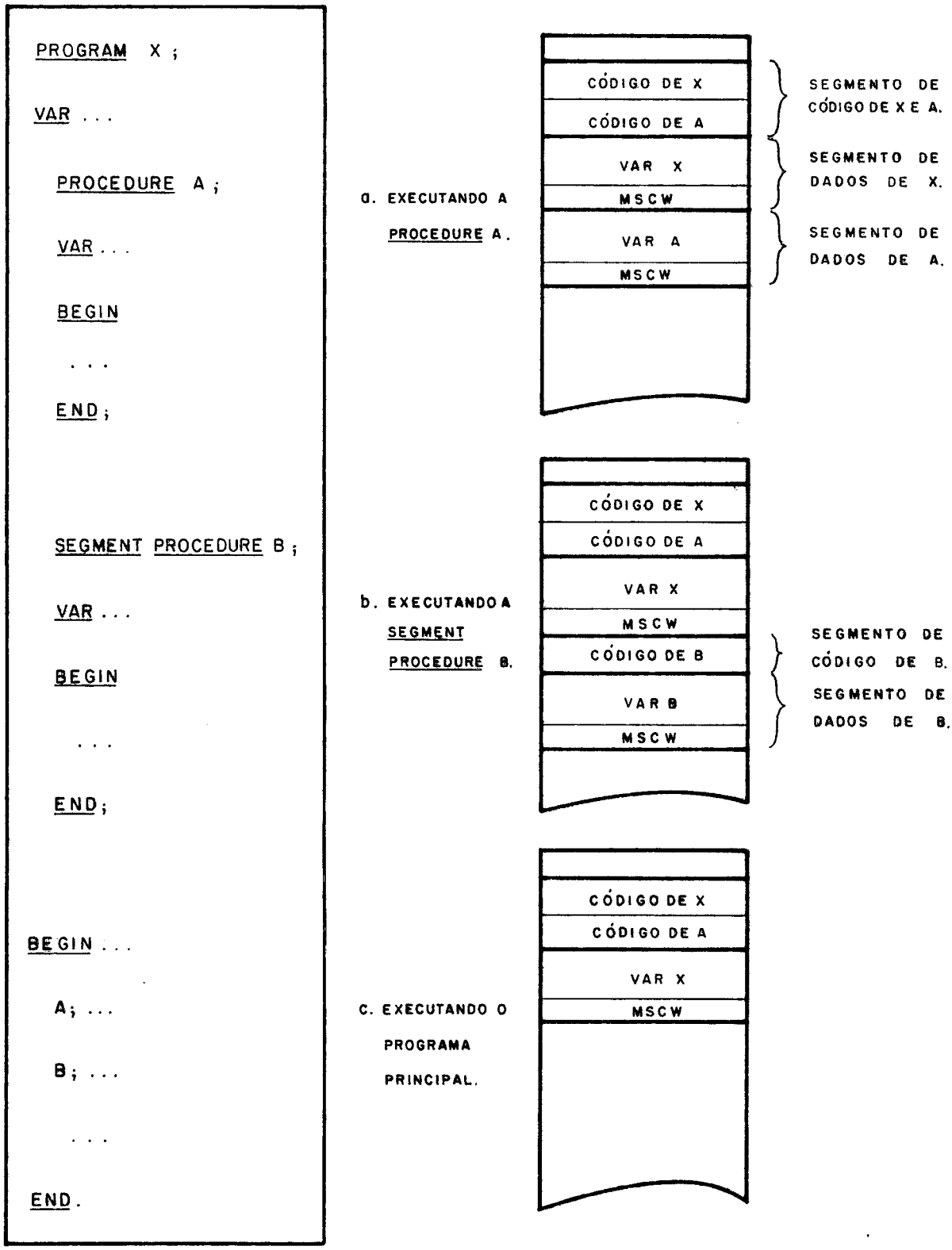


FIGURA IV. 4 - CONFIGURAÇÃO DO STACK EM ALGUNS INSTANTES DA EXECUÇÃO DE UM PROGRAMA.

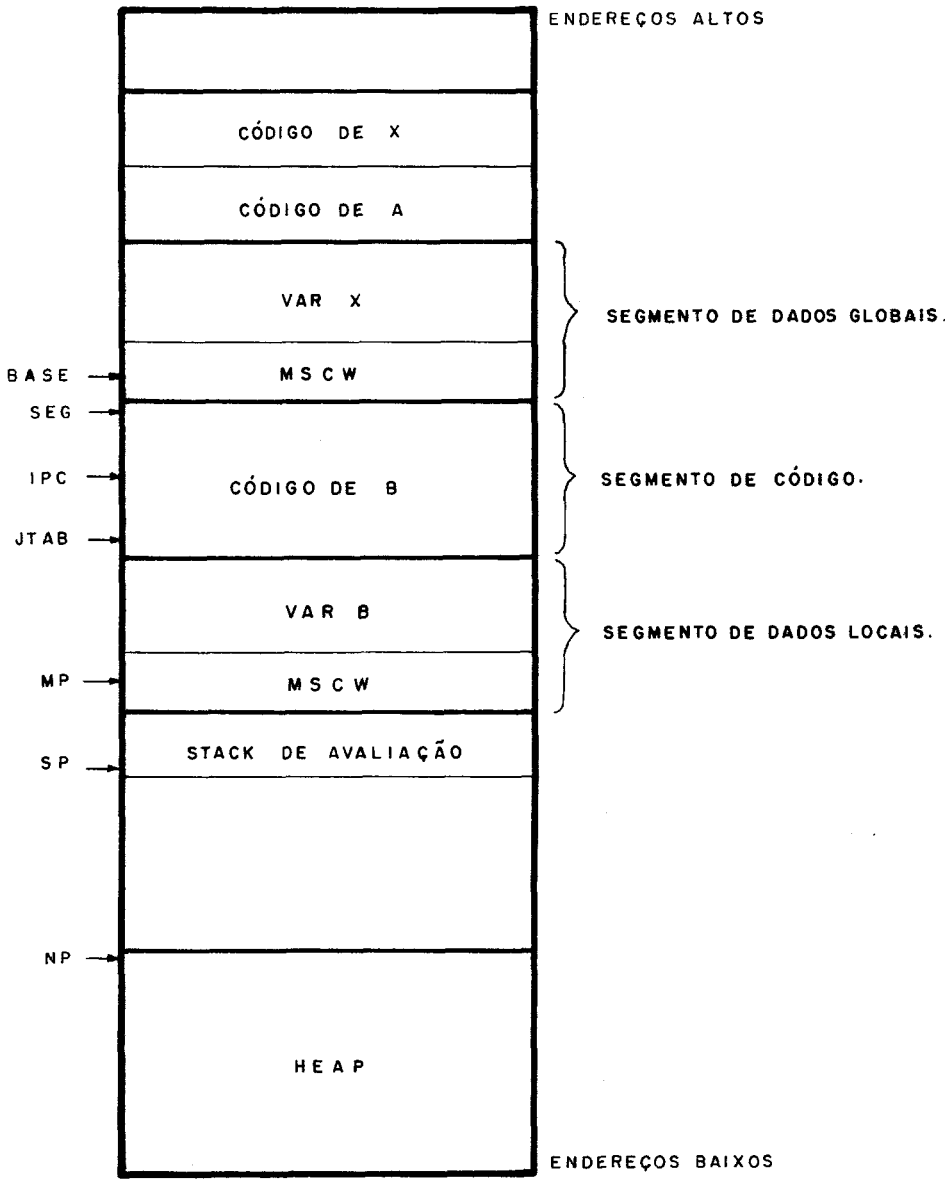


FIGURA IV. 5 - MAPA DA MEMÓRIA DURANTE A EXECUÇÃO DA SEGMENT PROCEDURE B.

IPC - contém o endereço do próximo p-code a ser executado dentro do segmento de código da procedure corrente;
BASE - aponta para o segmento de dados que contém as variáveis globais do programa;
MP - aponta para o segmento que contém as variáveis locais do programa. É o último segmento de dados formado no stack;
SP - aponta para o topo do stack corrente;
NP - aponta para o topo do heap corrente.

O "mark stack" ampliado da Figura IV.6 mostra os parâmetros da máquina-p que são armazenados durante a construção do segmento de dados de uma procedure. Os valores de MSSP, MSIPC, MSSEG, MSJTAB, MSDYN e MSBASE refletem o estado da máquina-p antes da ativação da nova procedure, correspondendo aos valores dos registros SP, IPC, SEG, JTAB, MP e BASE respectivamente. O valor de MSBASE só é armazenado no MSCW quando há alguma alteração registrada no valor do registro BASE, ou seja, durante a chamada a procedimentos globais. O campo MSSTAT aponta para o segmento de dados da procedure cujo nível léxico é imediatamente inferior (mais global) ao da procedure corrente. O encadeamento dos diversos valores de MSTAT constitui o "link" estático da máquina-p, enquanto que os diversos valores de MSDYN correspondem ao "link" dinâmico da mesma.

IV.2.1. TIPOS DE DADOS MANIPULADOS PELA MÁQUINA-P

As informações na máquina-p são, na maioria dos casos, alinhadas por palavras. As operações realizadas no topo do stack, por exemplo, assumem que os operandos ocupam ao menos o tamanho de uma palavra, mesmo que nem todas as informações da palavra sejam válidas. A única excessão à regra do alinhamento por palavra se verifica nos segmentos de código, pois tanto as instruções do código-p quanto alguns operandos ocupam apenas um byte. Os tipos de variáveis suportados pelo Sistema-p na versão I.5 são:

- Booleano: (1 palavra) O bit menos significativo indica o valor (falso = 0, verdadeiro = 1) da variável.
- Inteiro: (1 palavra) Valores representados no intervalo

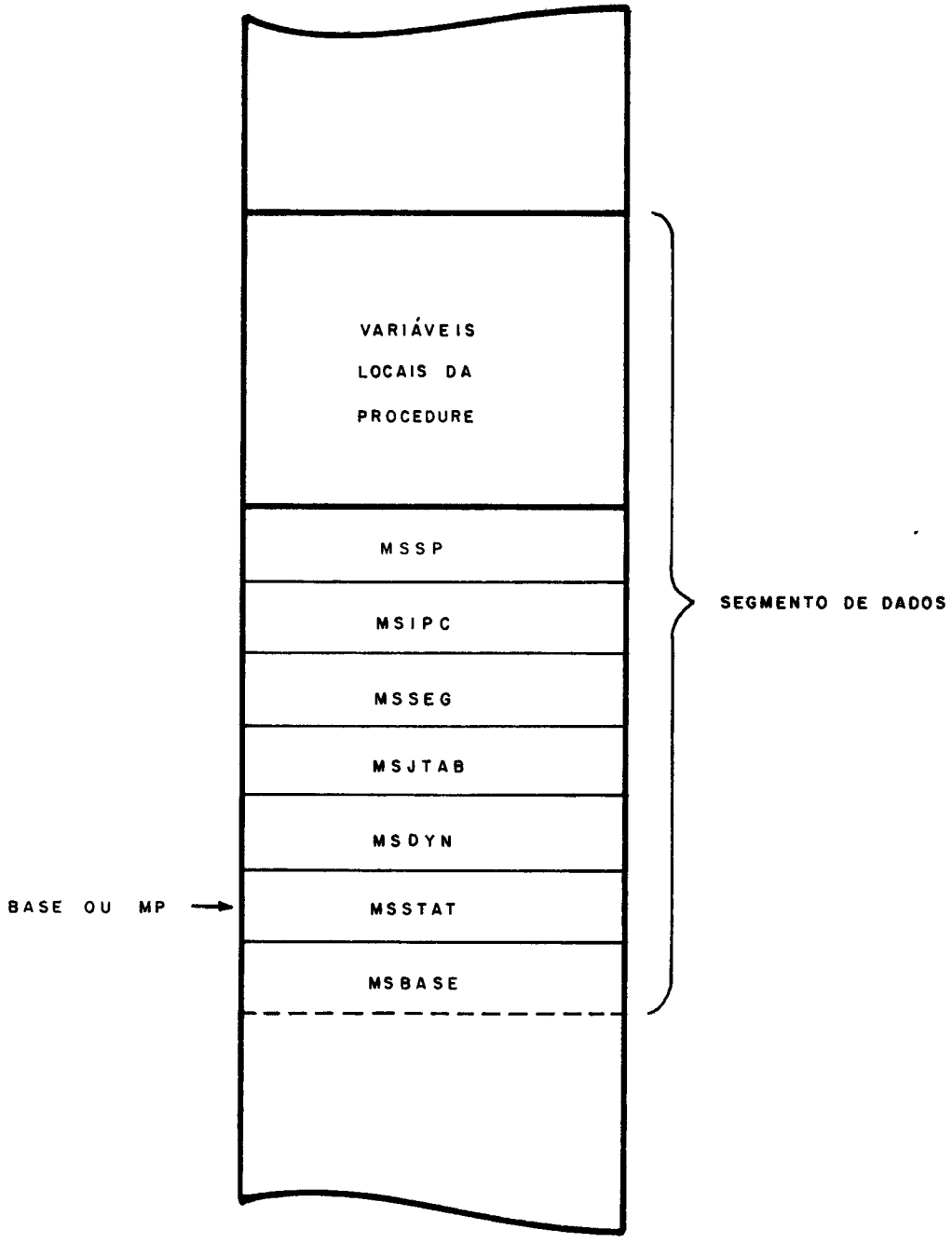


FIGURA IV. 6 - UM SEGMENTO DE DADOS AMPLIADO.

-32768..32767 em complemento a 2.

- Escalar: (1 palavra) Valores no intervalo 0..32767.
- Caracter: (1 palavra) O byte mais baixo contém o caracter ASCII.
- Real: (2 palavras) O formato é dependente da implementação. O sistema é arranjado de tal forma que só o interpretador precisa ter informações detalhadas sobre o formato interno dos números reais. No interpretador para o 8086 foi utilizado o padrão estabelecido pelo IEEE.
- Apontador: (1 ou 3 palavras) O tamanho depende do tipo do apontador. Para as estruturas usuais é empregada uma palavra apenas, mas para os campos internos das estruturas compactadas ("packed"), os apontadores ocupam 3 palavras: o apontador para a palavra que contém o campo, o tamanho do campo em bits e a indicação da posição inicial do campo dentro da palavra.
- Conjunto: (0 a 255 palavras no segmento de dados, 1 a 256 palavras no stack) Os conjuntos são implementados como vetores de bits, sempre com o índice inferior nulo. Quando o conjunto está no segmento de dados, todas as palavras contêm informações válidas. Quando está no stack, uma palavra é reservada para indicar a quantidade de informações úteis existentes no conjunto. Uma instrução de ajuste de tamanho (ADJ) deve ser executada, antes da transferência de um conjunto que esteja armazenado no stack para um segmento de dados.
- Registro e Matriz: (qualquer tamanho, com no máximo 16384 palavras em uma dimensão) As matrizes são armazenadas por coluna, com índice inferior nulo. Somente os campos e elementos são carregados no stack, nunca a estrutura completa. As matrizes compactadas devem conter um número inteiro de elementos em cada palavra, mesmo que hajam bits não aproveitados.
- "String": (1 .. 128 palavras) A implementação de "strings" visa a simplificação do uso de estruturas compactas de caracteres (packed array of char). Um string com n elementos ocupa (n div 2) + 1 palavras. O primeiro byte indica o tamanho corrente do string e o espaço restante contém o conjunto de caracteres válidos do mesmo.
- Constantes: É permitida a representação de constantes nos

tipos escalares, conjuntos, strings, inteiros e reais. As constantes podem estar inseridas no conjunto de instruções ou nos segmentos de dados.

IV.2.2. CONJUNTO DE INSTRUÇÕES

As instruções da máquina-p ocupam um ou dois bytes, que são seguidos de zero a quatro parâmetros. A maioria dos parâmetros especificam uma palavra de informação através de cinco tipos básicos:

- Byte sem sinal: O byte mais significativo do parâmetro é implicitamente considerado nulo.
- Byte com sinal: O bit de sinal do byte baixo é estendido para o byte mais significativo do parâmetro.
- Byte: Pode ser tratado de qualquer uma das duas maneiras acima, pois o seu valor está sempre no limite 0..127.
- Grande: Dependendo do seu valor, este parâmetro pode ocupar um byte (0..127) ou dois (128..32767). A indicação do tamanho está no bit 7 do primeiro byte. Se for zero, o parâmetro é pequeno, e os bits restantes contêm o seu valor. Caso contrário, os dois bytes são usados, com o primeiro deles sendo o mais significativo.
- Palavra: O significado dos 2 bytes deste tipo de parâmetro variam com sua aplicação.

A máquina-p versão I.5 contém mais de 200 instruções, que podem ser classificadas funcionalmente em cinco categorias:

- Busca, indexação, armazenamento e transferência de variáveis: São 77 instruções responsáveis pelas operações de movimentação de dados. Os operandos utilizados incluem constantes, endereços e variáveis, que podem estar situadas no topo do stack, nos segmentos de código, segmentos de dados estáticos (locais, globais e intermediários) ou dinâmicos.

- Operações aritméticas e lógicas no topo do stack: São previstas operações lógicas com variáveis inteiras, reais ,

binárias, conjuntos, strings, matrizes e registros. Os números inteiros e reais ainda podem ser usados em operações aritméticas diversas, num total de 84 códigos diferentes.

- Desvios: Além dos desvios condicionais (3) e do incondicional é previsto um tipo de desvio especial para os comandos do tipo "case".

- Ativação e retorno de procedimentos: Dentre as nove instruções deste grupo, existem quatro para ativar procedures situadas no mesmo segmento de código corrente e uma para aquelas que pertencem a um outro segmento. É previsto ainda um tipo especial de procedimento, que é implementado na linguagem nativa do processador real, chamada "standard procedure". O CSP ("call standard procedure") permite que sejam ativados, a partir da linguagem de alto nível, mais de 30 procedimentos diferentes, que estão associados ao controle das operações de e/s, ao tratamento de strings, à compilação de programas, etc..

- Operações de controle: Tratam-se de códigos que dizem respeito ao funcionamento da máquina de um modo geral. Apesar de alguma indefinição, na versão I.5 são previstas instruções para "breakpoints" (não implementadas), medição de tempo ("elapsed time"), parada ("halt"), etc..

V. ARQUITETURA PROPOSTA

V.1. PRINCIPAIS ASPECTOS CONSIDERADOS

Não se pretende impor restrições muito severas ao projeto, visto que é preciso verificar inicialmente a potencialidade das soluções apresentadas. Mesmo assim, será tomado um cuidado especial para que não se perca de vista os objetivos de mais longo alcance, em função dos quais desde já deverão ser considerados os seguintes pontos:

- Adequação ao processamento científico: Conforme já mencionado anteriormente, a máquina deverá ser projetada para atender a programas caracterizados por uma grande quantidade de operações em ponto flutuante, um estreito grau de interação entre os diversos procedimentos e um considerável volume de memória para armazenar os dados e códigos.

- Arquitetura baseada em microprocessadores: Pelos motivos já expostos no Capítulo III, deverá ser utilizada uma arquitetura baseada em microprocessadores. O primeiro requisito citado acima exigirá dos diversos processadores o acesso rápido a uma memória comum e aos principais periféricos do sistema . A performance final do conjunto deve ser comparável à dos computadores de médio porte existentes no mercado.

- Exploração do paralelismo: O baixo custo dos microprocessadores os tornam especialmente atraentes como recursos de computação. Sua utilização entretanto, fica às vezes restrita devido a fatores como baixa velocidade de processamento e limite da capacidade de endereçamento. A tendência apresentada pelas novas gerações destes componentes representam naturalmente um salto de qualidade nestes aspectos, mas o problema, em princípio, continua a existir para aplicações mais sofisticadas. É necessário organizar a forma de cooperação entre os vários elementos inteligentes da estrutura de forma que sejam obtidos níveis de eficiência e confiabilidade no sistema final.

V.2. TOPOLOGIA

A Figura V.1 apresenta uma visão geral da arquitetura proposta, composta por módulos interligados através de um barramento paralelo de alta velocidade. Os blocos mais importantes a serem considerados são as unidades de processamento, a memória global e os periféricos comuns do sistema.

O potencial alcançado pelo grupo de processadores da estrutura pode ser considerado equivalente ao de um (único) computador bastante rápido, com a vantagem de permitir que o sistema seja gradualmente degradável. A retirada de um processador apenas diminui o desempenho global da máquina, tornando mais lenta a CPU virtual equivalente.

Os itens que se seguem irão abordar cada um dos componentes da estrutura, permitindo que as alternativas e soluções consideradas no projeto sejam analisadas.

V.3. BARRAMENTO GLOBAL

A estrutura do barramento global é um dos elementos de maior importância em sistemas deste tipo, pois toda a performance da máquina está baseada no funcionamento eficiente do meio de interligação entre os módulos.

Por tratar-se de uma estrutura bastante flexível, que vem sendo largamente utilizada por diversos fabricantes, será adotado no projeto o padrão Multibus da Intel. Este barramento é composto por 24 linhas de endereço, 16 linhas bidirecionais de dados, 8 níveis de interrupção e alguns sinais para controle, temporização e alimentação. Este padrão, que foi objeto de estudo do IEEE, foi formalmente especificado através do registro IEEE-P796.

O espaço de endereçamento disponível prevê o acesso a 16 Mbytes de memória e a 4K portas de entrada/saída. O protocolo do

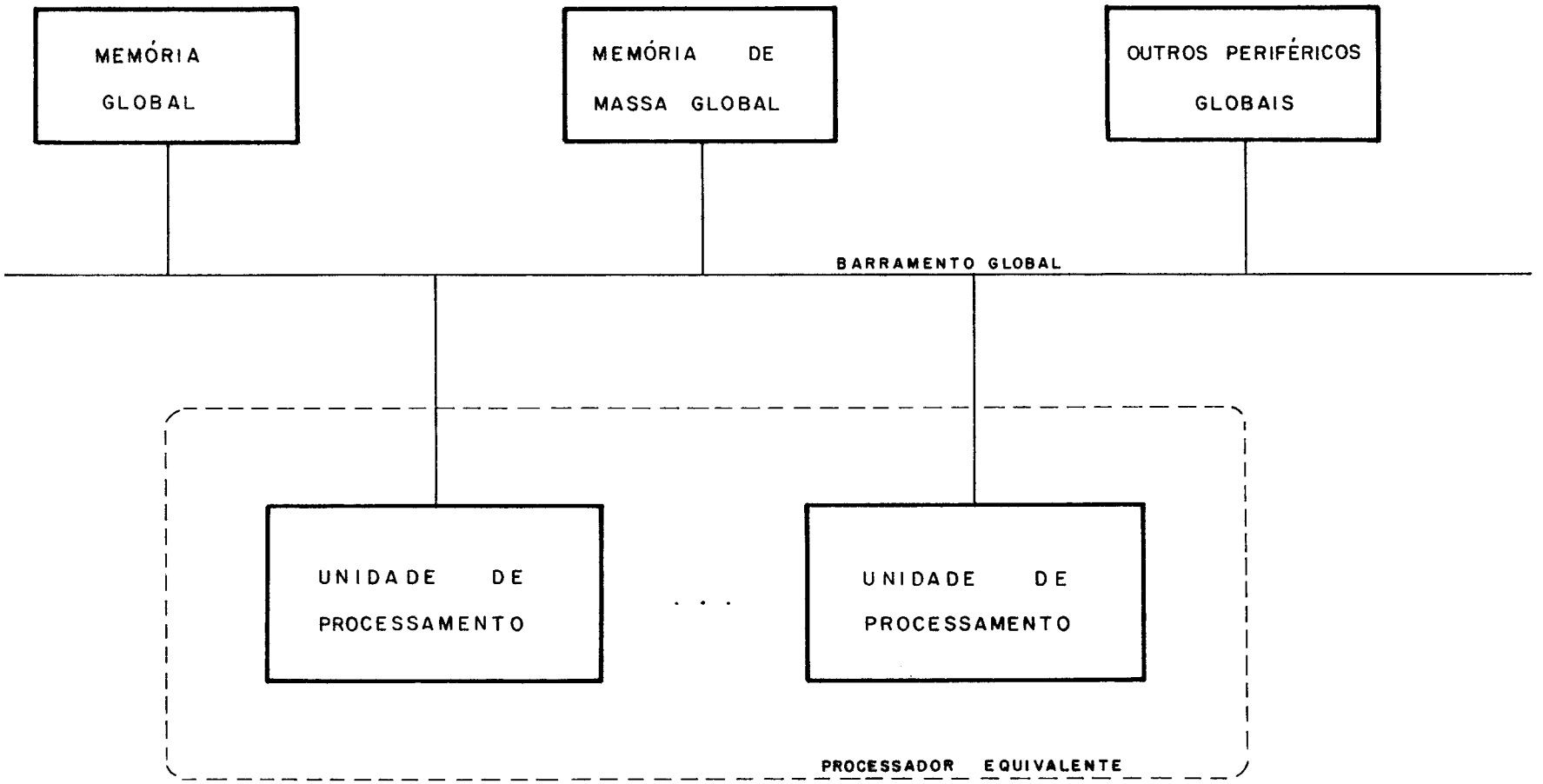


FIGURA V. 1 - ARQUITETURA PROPOSTA PARA O PROJETO.

barramento é baseado no conceito de mestre-escravo, onde o dispositivo que é mestre adquire total controle sobre o Multibus. Os dispositivos escravos decodificam as linhas de endereço e agem segundo os sinais de comando gerados pelos mestres. O Multibus apresenta ainda uma outra característica bastante desejável, que é a habilidade de suportar mais de um mestre no sistema, num total de até 16 módulos mestres.

Maiores detalhes sobre o funcionamento deste barramento estão disponíveis em manuais específicos sobre o assunto (62 e 63).

V.4. PROCESSADOR REAL

A Figura V.2 apresenta um detalhamento maior das unidades de processamento mostradas no início deste Capítulo. A estrutura proposta tende a minimizar a interferência do processamento local sobre a taxa de ocupação do barramento global, através da utilização de uma barra independente, que conecta a CPU a alguns recursos locais de memória e entrada/saída. Estas facilidades reduzem em parte a grande desvantagem das arquiteturas baseadas em barramentos multiplexados no tempo, que é a contenção registrada no sistema de interconexão dos módulos.

V.4.1. O MICROPROCESSADOR

As famílias de microprocessadores de 16 e 32 bits disponíveis atualmente oferecem uma quantidade razoável de componentes para implementação da arquitetura ora proposta. Com a tecnologia HMOS possibilitando o emprego de taxas de clock da ordem de 10MHz, com os conjuntos de instrução crescendo em quantidade e potência e com a densidade dos "chips" atingindo centenas de milhares de portas lógicas abre-se um amplo espectro de opções à disposição do projetista. Além dos processadores propriamente ditos, que já são bastante poderosos, estão sendo desenvolvidos circuitos auxiliares que tem aumentado ainda mais a potência de determinadas tarefas específicas. Estes circuitos são acoplados aos processadores de três formas: como simples periféricos

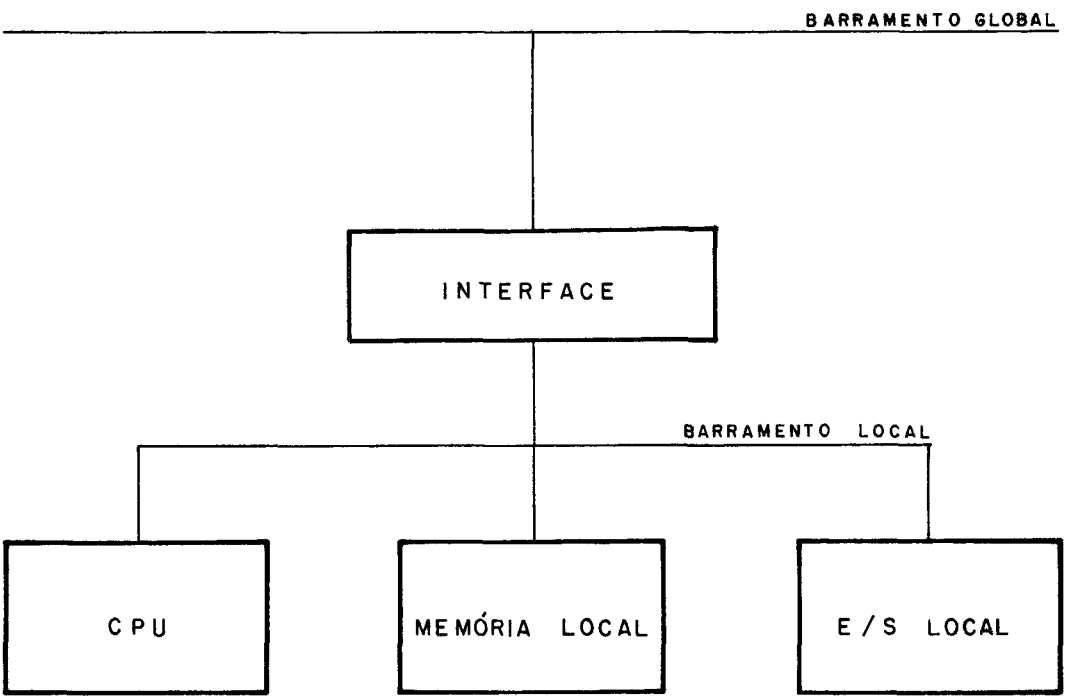


FIGURA V.2 - ARQUITETURA INTERNA DA UNIDADE DE PROCESSAMENTO.

programáveis; como processadores escravos que se comunicam por intermédio de um protocolo ou através de coprocessamento. Nesta última modalidade, o circuito auxiliar decodifica suas próprias instruções automaticamente, em paralelo com a CPU, ampliando efetivamente o conjunto de instruções originalmente suportado pelo processador principal.

A linha de componentes da Intel deverá ser adotada no projeto, em função principalmente da experiência acumulada no CEPEL com os produtos deste fabricante. Para minimizar os problemas de construção de hardware, serão inicialmente empregadas placas de circuito impresso já montadas e testadas. Isto apresentará a vantagem adicional de permitir uma reprodução em número limitado do equipamento desenvolvido.

O microprocessador APX86 tem capacidade de endereçamento para 1 Mbyte de memória, 64K portas de E/S, "clock" de até 10MHz e um conjunto poderoso de instruções, que inclui operações de multiplicação e divisão inteira, além de manipulação de "strings" em 24 modos de endereçamento. Além disso, ele pode ser configurado de duas maneiras distintas, que permitem ao usuário optar entre simplicidade de hardware ou uma maior performance.

Outros microprocessadores como o APX186 e o APX286 são evoluções naturais da linha APX86, sendo que este último representa um melhora significativa em relação aos microprocessadores de 16 bits atuais. O APX286, em um dos modos de operação, é seis vezes mais rápido que o APX86 , e em outro adiciona diversas funções ao "chip" da CPU. As características principais deste novo componente são descritas sumariamente a seguir (64):

- O espaço real de endereçamento foi estendido para 16 Mbytes de memória, e cada tarefa dispõe de 1 Gbyte de espaço virtual de endereçamento. Funções de gerenciamento e proteção de memória são suportadas, permitindo o desenvolvimento de sistemas complexos bastante confiáveis, a um custo reduzido.

- A arquitetura interna da CPU inclui quatro processadores independentes que, através de execução concorrente, otimizam o

acesso ao "bus" e aumentam o rendimento do componente como um todo.

- O hardware incorpora circuitos especiais que executam funções específicas de chaveamento de tarefas (18 us), atendimento a interrupções (3 us), divisão e multiplicação inteira (10 vezes mais rápido que o APX86). Existe também uma lógica para acelerar as funções de gerenciamento de memória (tradução de endereço lógico para físico, separação entre áreas do sistema e dos usuários, proteção de memória, etc.).

Aliada à capacidade intrínseca dos seus processadores, a Intel fornece componentes adicionais que permitem aumentar ainda mais a sua performance. É o caso dos coprocessadores que realizam funções específicas, como por exemplo o Processador de Dados Numéricos. O 8087, como é chamado, utiliza o padrão IEEE na manipulação de números inteiros de 32 e 64 bits e em ponto flutuante com 32, 64 e 80 bits. Este coprocessador numérico aumenta a capacidade para tratamento de números reais de um microprocessador APX86 em duas ordens de grandeza.

A placa de CPU que deverá ser inicialmente empregada no projeto é a SBC 86/12A (65), que está configurada com 64K bytes de memória RAM e um coprocessador de ponto flutuante 8087. A conexão da placa com o resto do sistema é realizada através da barra paralela Multibus, adotada no projeto.

Estão disponiveis no mercado outras familias de microprocessadores que também se enquadram nas características citadas no item V.1, e que poderiam igualmente ser empregadas no projeto. A seguir são apresentadas algumas destas famílias de microprocessadores.

A Motorola dispõe de um microprocessador com uma arquitetura interna de 32 bits armazenada em um invólucro "dual-in-line" com 64 pinos. Capaz de endereçar 16 Mbytes de memória sem nenhuma segmentação ou paginação, o MC68000 apresenta uma arquitetura totalmente microprogramada, que permite a inclusão de novas instruções, sem que haja necessidade de alterar o "chip" do

processador. Os registros internos tem 32 bits de largura e a comunicação com o exterior é feita através de um barramento assíncrono que inclui 16 linhas de dados e 23 linhas de endereço.

O conjunto de instruções do MC68000 contém apenas 56 comandos básicos, mas quase todos eles podem tirar vantagem dos 14 modos de endereçamento disponíveis. Estes comandos incluem multiplicação e divisão inteira, interrupções especiais, funções poderosas para ativação e retorno de procedures, além de outras instruções com múltiplas funções.

O gerenciamento de memória no sistema MC68000 pode ser realizado por um componente externo: o MC68451. As suas principais características (66) incluem tradução de endereços lógicos para físicos, proteção para os dados e programas do sistema e dos usuários, além de suporte para paginação e segmentação da memória.

A série Z8000 de microprocessadores da Zilog apresenta uma estrutura regular de registros, que é bastante apropriada para uma geração eficiente de código compilado. Além dos registros especiais do sistema, existem 16 registros de uso geral, que podem ser usados como acumuladores. Dentre estes últimos, 15 podem servir para indexar a memória. A arquitetura também prevê operações típicas de multiprocessamento através de linhas especiais de controle.

Para resolver os problemas de gerenciamento de memória em sistemas grandes, a Zilog apresenta duas alternativas. A primeira consiste em utilizar uma unidade especial (Z8010) que manipula 64 segmentos de tamanhos variados, mapeados em um espaço total de 16 Mbytes de endereçamento. Esta MMU ("memory management unit") além de realizar relocação dinâmica de segmentos e funções de proteção, é capaz de suportar sistemas multiprogramados com memória virtual. Uma outra solução para o problema de gerenciamento de memória seria a utilização da nova geração de processadores, o Z8003 e Z8004, que incorporam ao processador as características necessárias para implementação de

memória virtual. Esta nova geração é compatível com a anterior (Z8001 e Z8002), mas também incorpora diversas funções novas (67 e 68).

A família NS16000 da National, recentemente lançada, é composta por três membros: 16008, 16016 e 16032. São basicamente máquinas de dois operandos que possuem 4 modos de endereçamento. Estes modos são eficientemente explorados por um conjunto de mais de 100 instruções. Suas arquiteturas internas se baseiam em 33 registros, dos quais 16 são de uso geral. Nem todos os registros estão no "chip" da CPU, pois o sistema é composto por três componentes principais: a CPU, o gerente de memória e o processador de ponto flutuante.

V.5. MEMÓRIA PRINCIPAL

Até algum tempo atrás, o projeto da memória de um computador tinha que levar em conta que o seu custo representava uma parcela significativa dos recursos totais investidos. Esta preocupação se encontra bastante reduzida atualmente, podendo inclusive servir como elemento decisivo na análise dos compromissos entre o volume de memória, a velocidade de processamento e a complexidade dos circuitos existentes em um projeto. Outros fatores passaram a ter uma relevância significativa, como por exemplo a confiabilidade e a flexibilidade com que a memória deve ser administrada. A implementação de técnicas que desvinculam a memória física do espaço lógico de endereçamento tem merecido uma atenção especial dos pesquisadores desta área.

No presente projeto são empregadas duas técnicas bastante conhecidas: a segmentação e a paginação. A superposição destes dois conceitos numa única estrutura permite uma flexibilidade bastante grande no gerenciamento da memória global do sistema.

A segmentação permite que haja uma divisão lógica do espaço de endereçamento, através da identificação de módulos estanques de armazenamento. Podem ser separados com este tipo de divisão os

trechos de código correspondentes aos diversos procedimentos de um programa, algumas áreas de dados, etc..

A paginação garante uma boa distribuição da memória física entre os diversos segmentos, aumentando a capacidade de remanejamento das áreas residentes na memória principal.

No contexto deste trabalho, será utilizado o termo "pacote" ao invés de "segmento", pois este último ficará reservado para descrever as regiões de memória manipuladas pelos microprocessadores específicos que deverão implementar os pacotes aqui projetados. A correspondência entre "pacote" e "segmento" dependerá portanto dos componentes que vierem a ser utilizados no projeto. Se todas as operações previstas não forem implicitamente realizadas pelos processadores reais, poderão ser empregadas técnicas de simulação por software ou até mesmo a inclusão de hardware para preencher os requisitos especificados.

Os pacotes se caracterizam por um conteúdo homogêneo de informações (código, dados estáticos, etc.) e por um tamanho totalmente variável, conforme mostra a distribuição típica da Figura V.3. Neste exemplo, os três pacotes existentes representam coleções distintas de dados. Cada um destes dados é acessado através de duas informações básicas: a identificação do pacote e o deslocamento dentro do mesmo. A Figura V.4 ilustra o acesso a um dado através de um apontador.

De acordo com o conteúdo do pacote, com as operações que nele serão realizadas e com o privilégio outorgado às tarefas em execução, o processador pode dispensar um tratamento especial às diversas regiões alocadas na memória. Para isto, é necessária a criação de um descritor que deverá conter as informações associadas a cada pacote. No presente trabalho, o descritor de um pacote deverá incluir os campos definidos pelo registro abaixo, que são em seguida detalhados:

```
type pacrec = record
                base: baserange;
                tamanho: offsetrange;
```

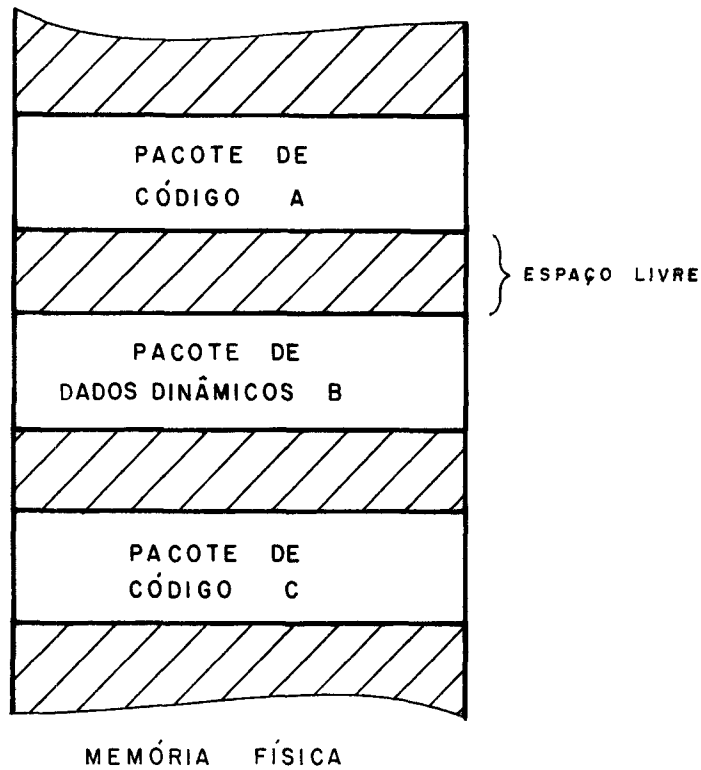


FIGURA V. 3 - DISTRIBUIÇÃO TÍPICA DOS PACOTES NA MEMÓRIA.

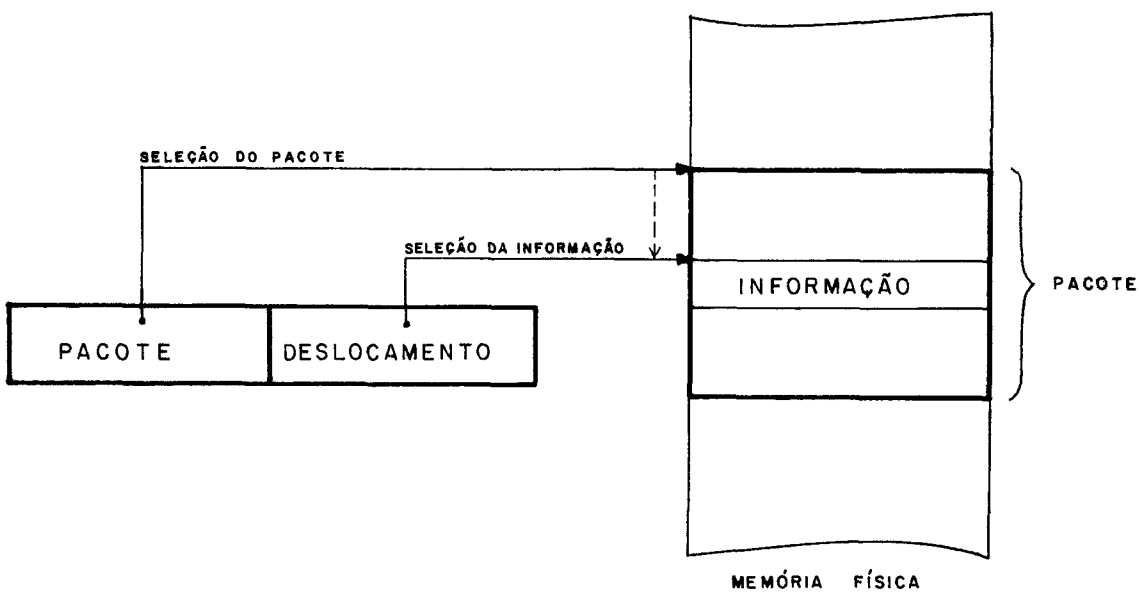


FIGURA V. 4 - ACESSO ÀS INFORMAÇÕES DE UM PACOTE.

```
        present: boolean;
        unidade: unitnum;
        bloco: integer;
        acesso: (noaccess, rdaccess, wraccess,
                 allaccess);
        tipo: pactype;
        wrback: boolean;
        inicpag: pagrange;
        util: integer
      end;
```

- A base representa a posição inicial do pacote no espaço real de endereçamento;
- O tamanho indica o deslocamento máximo que pode ocorrer dentro do pacote;
- O "flag" present serve para sinalizar a presença do pacote na memória física. Se for falso, uma interrupção de "pacote ausente" irá ocorrer, assim que houver o acesso ao mesmo.
- Os campos unidade e bloco contém as referências necessárias para buscar um pacote ausente na memória de massa do sistema.
- O acesso indica os tipos de operação que podem ser realizadas no trecho de memória delimitado pelo pacote, através das possíveis combinações das operações de leitura e escrita na memória.
- O conteúdo do pacote é descrito no campo tipo, que pode indicar um pacote de código, de dado ou até mesmo um espaço livre.
- Após a realização de uma operação de escrita em um pacote, o "flag" de wrback será tornado verdadeiro, indicando que se houver uma cópia na memória de massa, ela estará precisando ser atualizada.
- Os campos inicpag e util serão descritos mais adiante, no Capítulo VI.

Os descritores serão agrupados em uma tabela, que passa a fazer parte da área do sistema operacional:

```
var pactbl: array [pacrange] of pacrec;
```

O processador virtual deverá realizar para cada acesso à memória a seguinte sequência de operações:

- Através do número do pacote, buscar em pactbl o descritor do mesmo. A tabela é localizada por meio de uma referência física absoluta.
- Verificar se não há nenhuma violação das regras de acesso. Neste passo, devem ser utilizados os campos de tamanho, acesso e tipo. Em caso de falha, uma interrupção de "erro de acesso" deve ser gerada.
- Verificar se o pacote está presente na memória física, sinalizando a sua ausência com uma interrupção. Um procedimento especial irá carregar o pacote na memória física, permitindo que o processamento continue.
- Computar o endereço físico através do endereço base do pacote e do deslocamento fornecido pelo apontador que está sendo processado.
- Realizar a operação de acesso, atualizando o "flag" de wrback se for o caso.

Estas funções deverão ser preferencialmente executadas por hardware, para que não haja um retardamento excessivo no processamento. Cabe aqui ressaltar que nos microprocessadores mais modernos já são previstas estas funções (64 a 68).

Observando-se os procedimentos acima, conclui-se que um determinado pacote pode ser carregado em qualquer posição da memória física, desde que seja respeitada a sua contiguidade e o valor do campo base do descritor seja corretamente preenchido. Para exemplificar como se pode tirar partido desta facilidade, considere-se a situação mostrada na Figura V.5 em que o pacote X deve ser carregado na memória. O maior espaço físico consecutivo disponível não é suficiente para comportá-lo, o que vai obrigar a realização de uma compactação na memória. Após a compactação, que deve compreender a atualização das bases dos pacotes deslocados, o pacote X pode então ser carregado. A Figura V.6 ilustra a disposição final dos pacotes na memória.

Para realizar a compactação, no exemplo acima, foi necessário

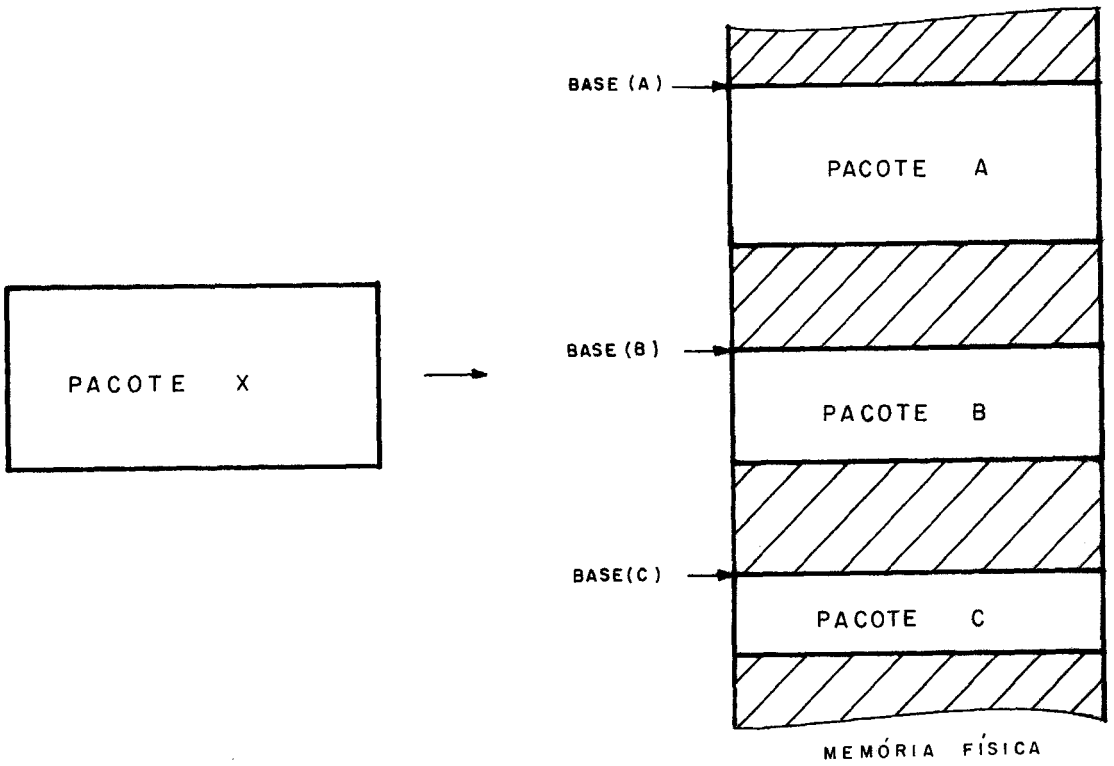


FIGURA V. 5 - CARREGAMENTO DE UM PACOTE: NÃO TEM ESPAÇO.

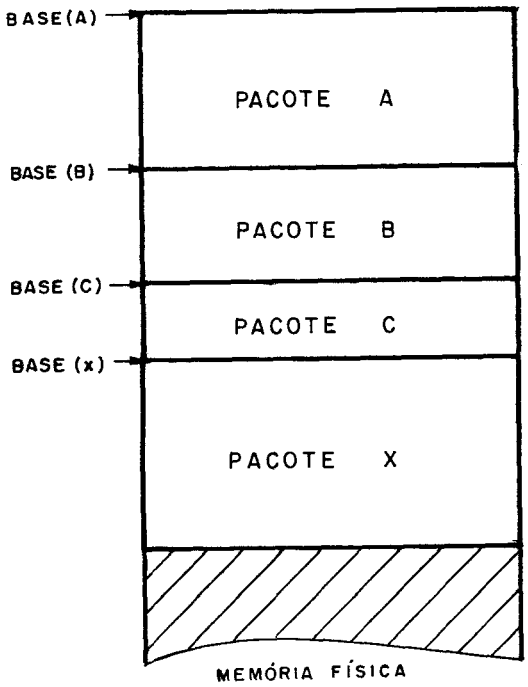


FIGURA V. 6 - PACOTE CARREGADO NA MEMÓRIA

deslocar os pacotes A, B e C. Entretanto, esta alternativa seria muito dispendiosa, se fosse realizada através de uma cópia, porque implicaria em uma transferência palavra e palavra do conteúdo dos pacotes envolvidos. Para resolver este problema, será incorporado aos pacotes de informação o conceito de paginação. A configuração da memória principal que permite a implementação destas novas funções está apresentada na Figura V.7. Esta estrutura se limita à memória conectada diretamente ao barramento principal do sistema, não estando incluídos os módulos de memória local,que podem ser acoplados a cada unidade de processamento (vide Figura V.2).

O conteúdo da memória RAM conectada à parte alta da barra de endereços permite controlar a divisão da memória física em páginas. O efeito resultante garante uma continuidade aparente da memória sem que as páginas estejam obrigatoriamente contíguas. A mesma operação de compactação realizada anteriormente pode agora ser realizada através de alterações na matriz de páginas, com a vantagem de que toda a página é deslocada com apenas uma cópia de palavras.

Um detalhe interessante desta configuração é a manutenção da compatibilidade com as memórias convencionais. Sob o ponto de vista do barramento global, a placa de memória continua a se comportar como antes, a menos de uma lógica de controle (acessada por E/S provavelmente), que irá programar a matriz de páginas.

A Figura V.8 mostra uma projeção do espaço virtual de endereçamento na memória real, com a divisão do pacote de informação em páginas. A Figura V.9 apresenta o processo de tradução do endereço virtual, após a introdução da matriz de páginas.

A diferença é que agora o campo base da tabela de pacotes serve para indicar a página inicial correspondente ao pacote na matriz de páginas. A transformação do endereço utiliza também uma parte do deslocamento do endereço virtual para localizar a página a ser acessada dentro do pacote.

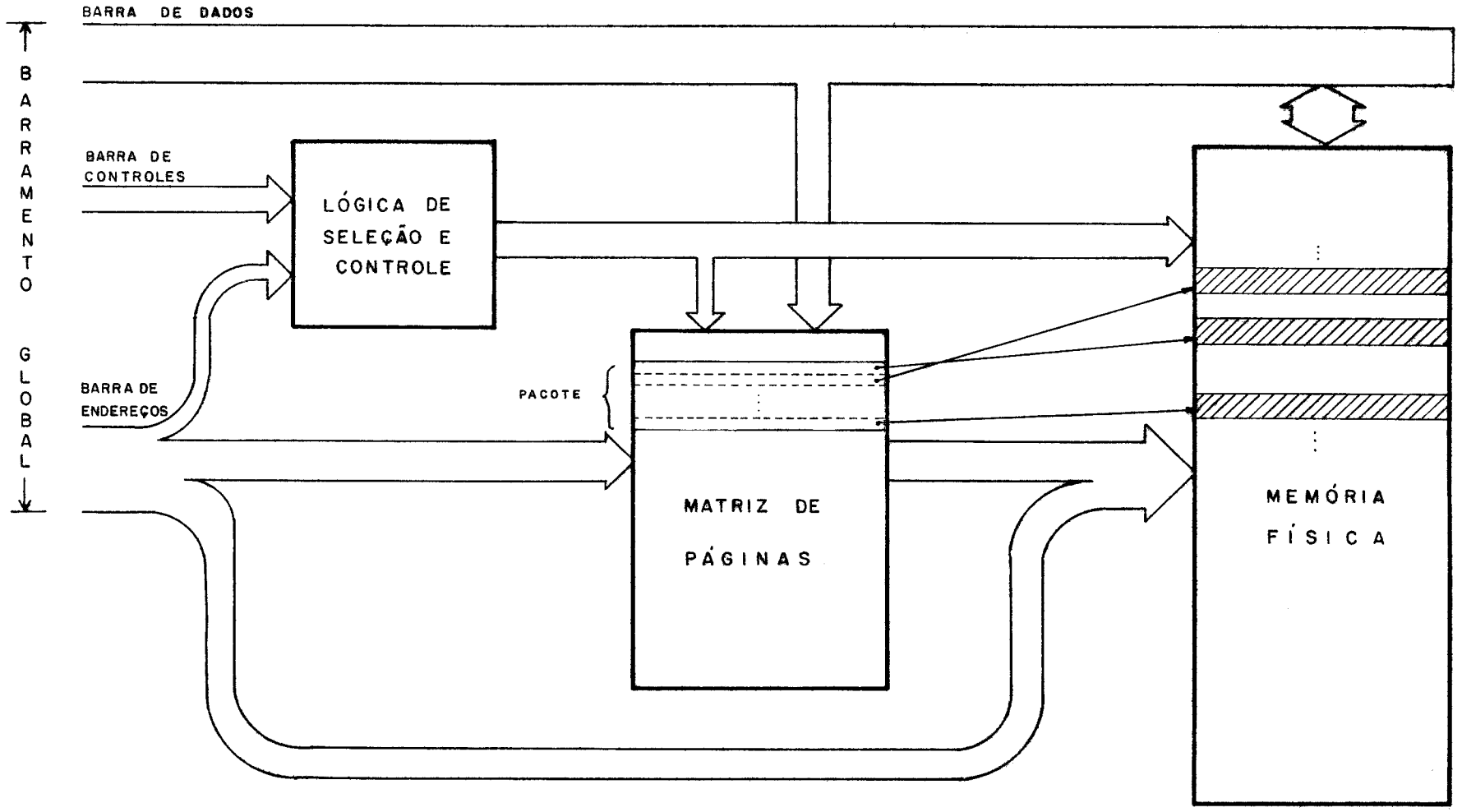


FIGURA V.7 - ESTRUTURA DA MEMÓRIA PRINCIPAL, DIVIDIDA EM PÁGINAS.

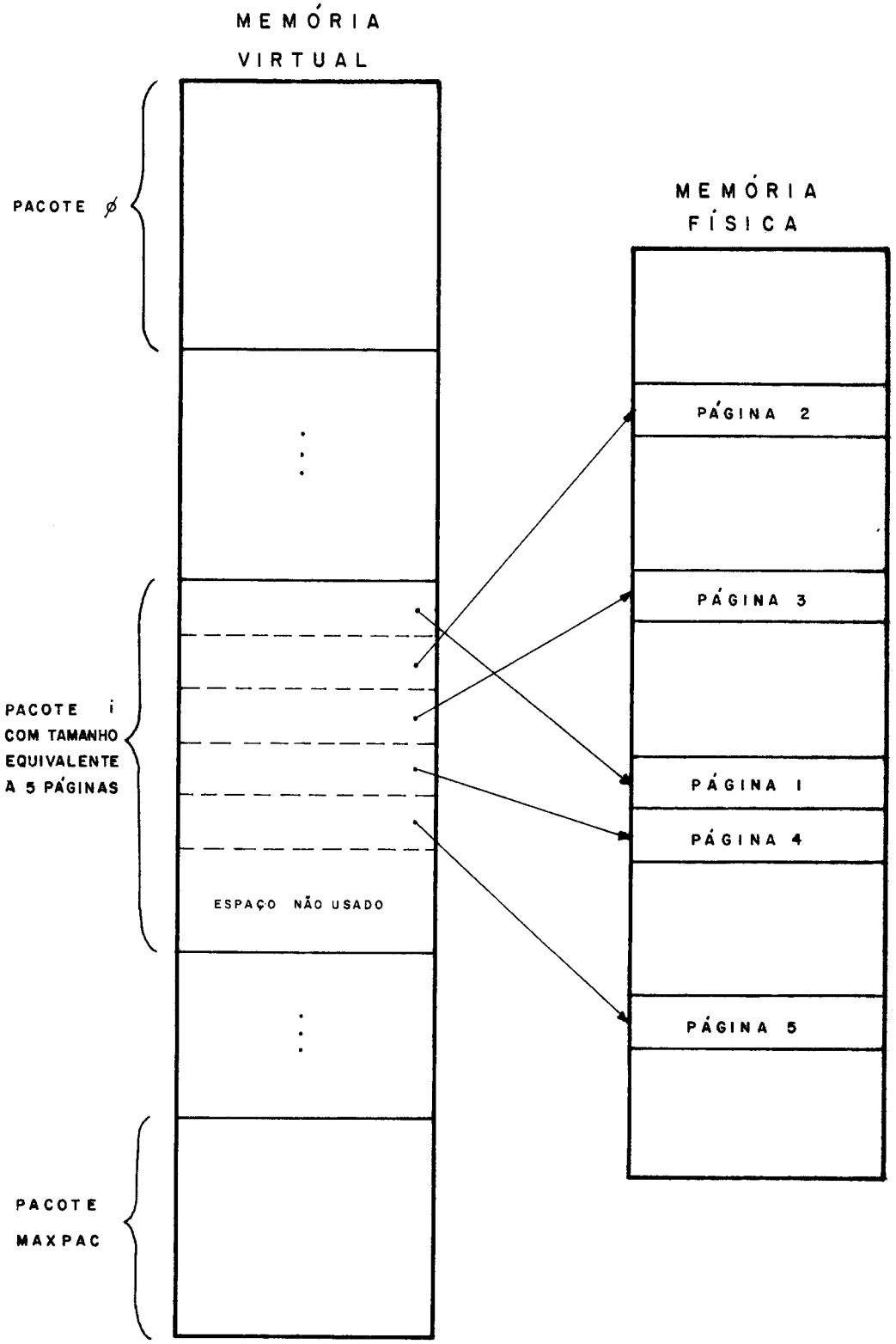


FIGURA V.8 - MAPEAMENTO DO ESPAÇO VIRTUAL DE ENDEREÇAMENTO NA MEMÓRIA FÍSICA

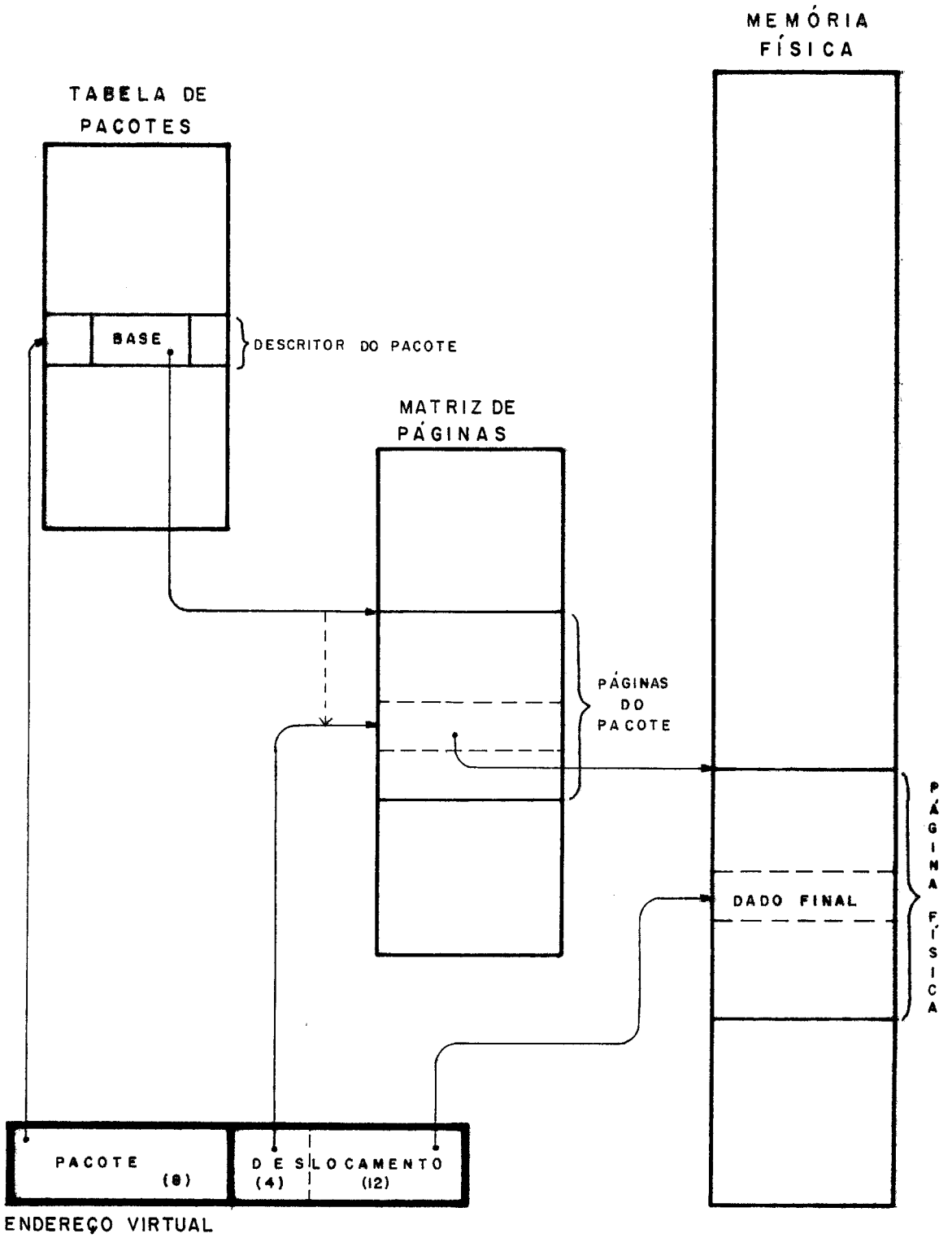


FIGURA V. 9 - CONVERSÃO DO ENDEREÇO VIRTUAL PARA A MEMÓRIA FÍSICA.

A programação da matriz de páginas é realizada através da procedure setpag, que recebe como parâmetros o endereço do registro a ser programado e o seu valor, conforme a descrição que se segue.

```
procedure setpag (reg: regrange, dado: regvalue);
begin
  matriz-de-páginas [reg]:= dado
end;
```

V.6. OUTROS RECURSOS DISPONÍVEIS

Sob o ponto de vista funcional, os demais recursos existentes na arquitetura podem ser agrupados em duas classes. A primeira delas é representada por elementos com características globais, da qual fazem parte os dispositivos destinados a acionar os principais periféricos do sistema. A segunda abrange as partes que fornecem facilidades locais às unidades de processamento, como é o caso de alguns periféricos que realizam operações internas. Enquanto o primeiro grupo é composto por elementos conectados diretamente ao barramento global, o segundo faz parte dos circuitos que são incorporados a todas as unidades de processamento existentes no sistema.

V.6.1. DISPOSITIVOS GLOBAIS DE E/S

Fazem parte deste grupo a memória de massa do sistema e os periféricos que realizam a comunicação com o elemento humano (impressoras, terminais de vídeo, etc.). Serão utilizados, sempre que possível, módulos inteligentes como o da Figura V.10 para controlar os periféricos globais do sistema. As principais razões que orientam esta escolha são:

- A programação dos periféricos através de um protocolo mais inteligente tende a reduzir a taxa de utilização do barramento global.

- As tarefas mais específicas de tratamento de informações e controle de acesso aos periféricos saem da área de responsabilidade das unidades processadoras, tornando o software básico menos sensível aos dispositivos de hardware incorporados ao sistema.

- A conexão do módulo de controle com o periférico apresenta um elevado grau de liberdade, permitindo ao projetista optar por um número bastante grande de técnicas de implementação de módulos controladores de entrada/saída.

Estão disponíveis no mercado excelentes alternativas para implementação destes controladores inteligentes. Além dos microcomputadores "single-chip", podem ser usados nesta tarefa alguns componentes que estão sendo especialmente desenvolvidos para exercer funções de controle sobre determinados periféricos.

As primitivas básicas para acesso aos dispositivos de E/S globais são, na maioria dos casos, as procedures unitread e unitwrite:

```
procedure unitread (unitnum,bufptr,bufsize,bloco,nowait);

procedure unitwrite (unitnum,bufptr,bufsize,bloco,nowait);
```

O valor de unitnum contém a unidade lógica a ser acessada;

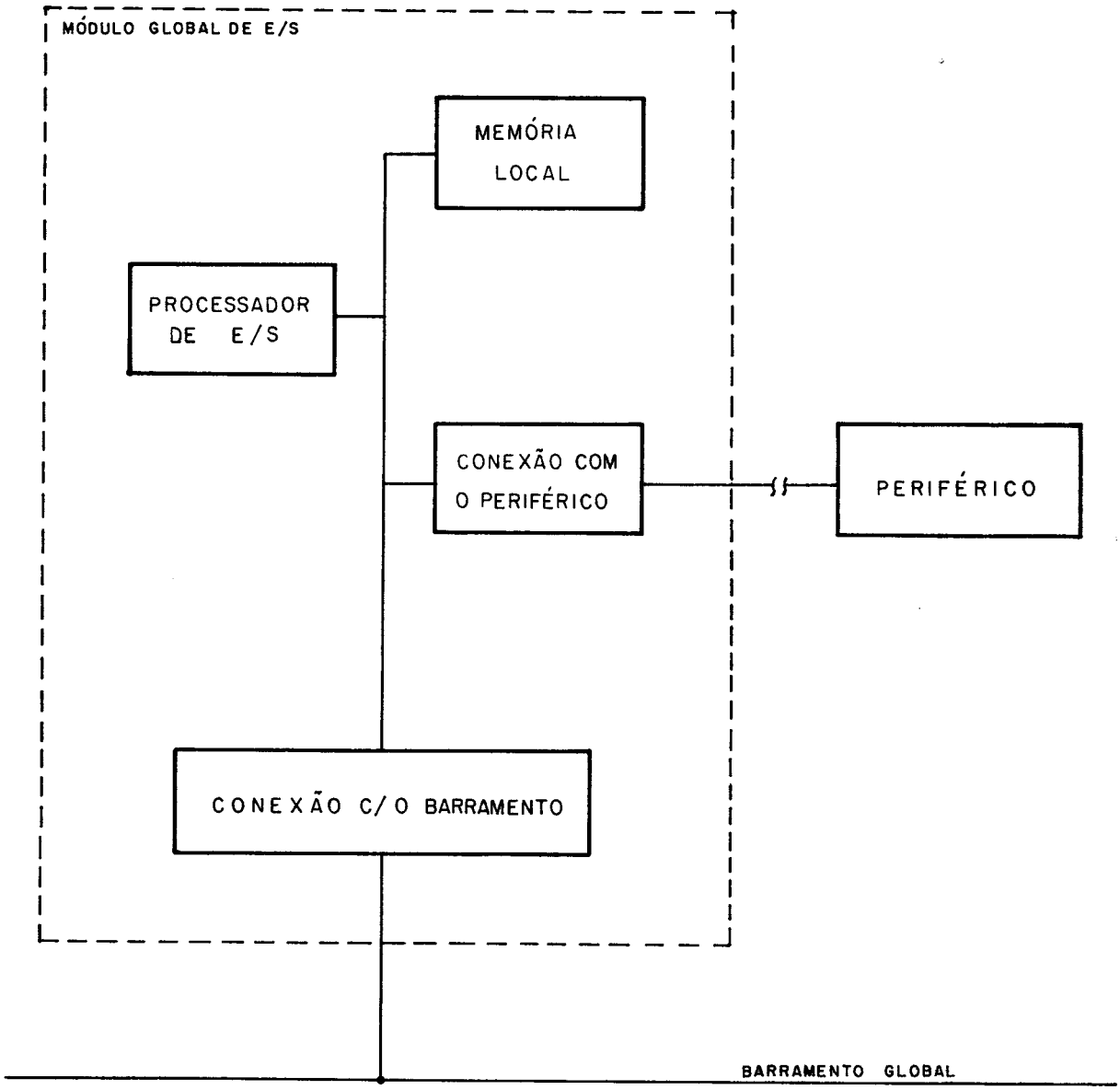


FIGURA V. 10 - DIAGRAMA TÍPICO DE UM MÓDULO GLOBAL DE E/S .

bufptr e bufsize descrevem o espaço de memória que deverá ser utilizado na transferência; o parâmetro bloco indica o número do bloco inicial a ser lido/escrito (em operações com o disco) e nowait é um "flag" usado para sinalizar operações assíncronas.

V.6.2. DISPOSITIVOS LOCAIS DE E/S

A necessidade de descentralização de algumas funções lógicas na arquitetura sugere que sejam introduzidas algumas facilidades locais nas unidades de processamento. Entretanto, deve-se tomar cuidado na especificação destas funções, pois um erro de avaliação pode gerar uma replicação desnecessária de recursos em todas as CPU's do sistema.

Dentre os periféricos que fazem parte deste grupo (temporizadores, lógica de deteção de falhas, interface com painel, etc.) é conveniente ressaltar o Controlador de Interrupções. Este periférico faz parte dos mecanismos de tratamento dos eventos que ocorrem no sistema. Os eventos são originados para sinalizar o fim de uma operação de E/S, a ocorrência de uma situação anormal, a passagem de um certo intervalo de tempo, etc. O sinais correspondentes a estes eventos podem ser originados de fontes globais (periféricos globais, falha na alimentação, etc.) ou locais (periféricos locais, etc.).

As fontes locais são tratadas individualmente por cada processador, enquanto que as interrupções globais são gerenciadas por uma lógica de distribuição, que deverá estabelecer, segundo uma política de prioridades, qual(is) processador(es) deverá(ão) ser responsável(is) pelo tratamento de um determinado nível de interrupção. Isto é implementado através da inibição dos níveis de interrupção que não estiverem associados a determinado processador, num determinado instante.

VI. SOFTWARE BÁSICO

Este capítulo contém a especificação de uma parte do Sistema Operacional que deve revestir a arquitetura apresentada no Capítulo V. De uma forma geral, as funções aqui incluídas estão mais intimamente ligadas à arquitetura, como a interpretação do código intermediário e a implementação da memória virtual.

Os procedimentos relativos ao escalamento, sincronismo e comunicação entre tarefas estão descritos em Barbosa et al (69) e serão sucintamente incluídos no texto, sempre que um esclarecimento se fizer necessário.

Inicialmente são propostas algumas adaptações na máquina-P que está descrita no Capítulo IV. O objetivo da máquina estendida é permitir que o Sistema Pascal UCSD possa usufruir dos recursos disponíveis na arquitetura, quais sejam, uma quantidade de memória maior que os 64K bytes originalmente previstos e a possibilidade de execução em diversos processadores.

Na segunda parte do Capítulo, serão apresentados os algoritmos que deverão realizar as funções relativas à implementação da memória virtual na máquina estendida. A linguagem de especificação escolhida foi o Pascal.

Finalmente, o Capítulo encerra com a especificação de um supervisor com capacidade para suportar vários programas independentes. A sua programação utiliza as ferramentas desenvolvidas neste trabalho e em Barbosa et al. (69).

VI.1. MÁQUINA-P ESTENDIDA

Um dos passos necessários à expansão da máquina-P é a incorporação de algumas ferramentas à linguagem Pascal UCSD que permitam o desenvolvimento de programas concorrentes. Estes mecanismos compreendem:

- comandos cobegin/coend para criação dinâmica e sincronismo entre tarefas;

- operações P/V em semáforos binários para assegurar a mútua exclusão no acesso a regiões críticas e o sincronismo em alguns casos especiais;

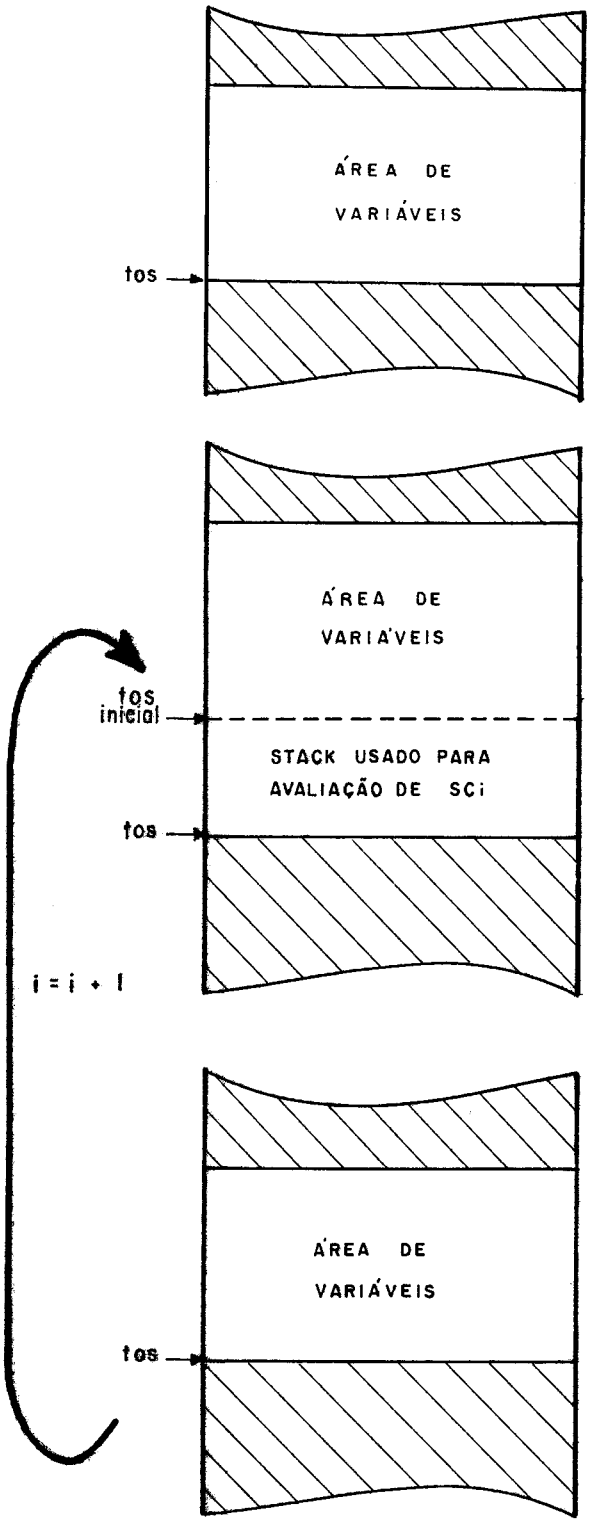
- operações wait/signal em filas de condição, que permitam o sincronismo dentro de regiões críticas.

Segue-se uma análise de alguns destes construtos, particularmente orientada para os tópicos mais importantes do projeto que será apresentado.

Considere-se inicialmente uma sequência de comandos da linguagem Pascal, onde as palavras reservadas begin/end delimitam um comando composto formado pelos subcomandos SC1, ..., SCn:

```
begin
 SC1;
 SC2;
  .
  .
  .
 SCn
end;
```

A Figura VI.1 ilustra o comportamento do stack da máquina-P UCSD durante a sua execução. A área situada acima do topo do stack inicial contém as variáveis estáticas já alocadas no programa. A região livre do stack, abaixo do topo, é reservada para armazenar as informações necessárias à avaliação dos subcomandos, e vai sendo preenchida sob demanda. Esta região pode, por exemplo, conter outra área para armazenamento de variáveis estáticas, se o subcomando incluir a chamada a algum procedimento (procedure ou function), ou então ser usada como simples espaço temporário de armazenamento, durante a avaliação de uma expressão aritmética. O importante a ressaltar é que cada subcomando SCi corresponde a um eventual crescimento e um



a - DISPOSIÇÃO DO STACK NO INÍCIO DO COMANDO COMPOSTO
tos : TOPO DO STACK.

b - DURANTE A EXECUÇÃO DO SCi O TOPO DO STACK É USADO PARA O ARMAZENAMENTO DE INFORMAÇÕES.

c - NO FINAL DA EXECUÇÃO DO SUB-COMANDO, O TOPO DO STACK RETORNA À POSIÇÃO ORIGINAL, E O PRÓXIMO SUBCOMANDO PODE SER EXECUTADO.
ESTA TAMBÉM É A SITUAÇÃO NO FINAL DO COMANDO COMPOSTO.

FIGURA VI.1 - STACK DA MÁQUINA-P UCSD DURANTE A EXECUÇÃO DE UM COMANDO BEGIN / END.

posterior retorno do topo do stack à sua posição original.

Analogamente, a sequência de processamento associada a um comando cobegin/coend irá resultar na execução dos subcomandos CC1, CC2, ..., CCn:

```
cobegin
 CC1;
 CC2;
  .
  .
  .
 CCn
coend;
```

A diferença registrada neste caso é que o tratamento dos subcomandos não será mais sequencial, e sim concorrente. A Figura VI.2 mostra esquematicamente esta nova situação, onde a área de variáveis situada acima do topo do stack inicial pode ser acessada por qualquer um dos subcomandos CCi, constituindo portanto uma região bastante adequada para comunicação entre os subcomandos concorrentes.

Na presente implementação, cada subcomando CCi deverá corresponder a uma chamada de procedure. Esta restrição viabiliza a introdução dos comandos cobegin/coend, sem que haja necessidade de alterações no compilador, bastando que sejam controlados alguns mecanismos de ativação de procedimentos existentes no interpretador.

A relação entre a área de variáveis comuns e o programa em execução está de acordo com a regra de escopo da linguagem Pascal, pois um determinado procedimento tem acesso a todas as variáveis declaradas nos níveis léxicos inferiores ao do próprio procedimento. Estas áreas de memória estão localizadas em registros de ativação situados acima do topo do stack. A ramificação do stack linear original criou um caminho para cada subcomando concorrente poder acessar as variáveis a que ele tem direito na estrutura.

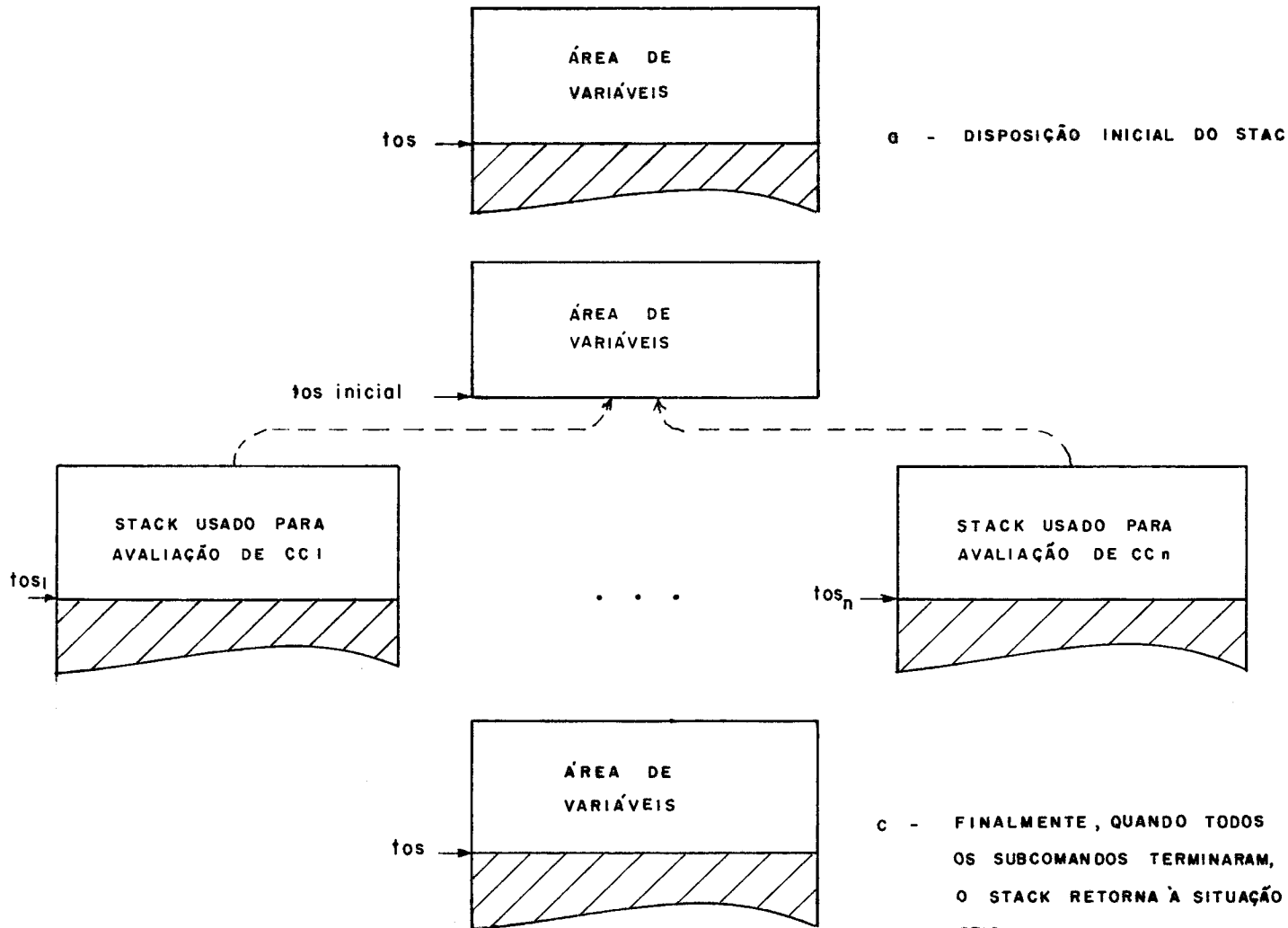


FIGURA VI. 2 - STACK DA MÁQUINA EXTENDIDA DURANTE A EXECUÇÃO DE UM COMANDO COBEGIN / COEND.

No caso mais geral, em que o stack irá se configurar segundo uma estrutura em árvore, as folhas corresponderão aos registros de ativação dos subcomandos mais recentes, já que a regra de funcionamento do cobegin/coend estipula que um comando composto só é dado por encerrado após a completa execução de todos os seus subcomandos.

Para concatenar a ramificação do stack com o conceito de memória virtual, ou mais especificamente, com os pacotes de informação citados no Capítulo V, será necessário introduzir inicialmente mais uma modificação na máquina-P original. Conforme foi definido naquele capítulo, os pacotes deveriam conter informações homogêneas, ou seja, as regiões de código e dado seriam alocadas separadamente. Sendo assim, o stack não mais será utilizado para armazenar código, ficando exclusivamente restrito aos registros de ativação das variáveis estáticas declaradas no programa. As funções de gerência de memória da máquina estendida irão alocar os trechos de código em um espaço virtual independente, na razão de um pacote de memória virtual para cada segment procedure declarada no programa. Esta implementação apresenta algumas vantagens em relação à máquina original, em virtude da flexibilidade adicional conseguida na manipulação dos programas residentes na memória. Anteriormente só havia previsão de carregamento de segment procedures no próprio stack, em tempo de execução.

Resta agora analisar como será mapeado o stack estendido no espaço virtual de endereçamento. O Algoritmo VI.1, empregado como exemplo, servirá para mostrar os principais aspectos a serem considerados neste problema.

O programa concorrente A ativa três procedimentos B, C e D. O procedimento C, por sua vez, ativa dois outros procedimentos E e F. O diagrama da Figura VI.3 representa uma situação típica em que B, D, E e F estão sendo executados concorrentemente. Existe uma correspondência direta entre os ramos da árvore formada e os processos que deverão concorrer pelos recursos existentes na máquina.

```
concurrent program a;
var ..
  concurrent procedure b;
  var ..
  begin..end;
  concurrent procedure d;
  var ..
    procedure d1;
    var ..
    begin .. end;
  begin .. end;
  concurrent procedure e;
  var ..
    procedure e1;
    var ..
    begin .. end;
    procedure e2;
    var ..
    begin .. end;
  begin .. end;
  concurrent procedure f;
  var..
  begin..end;
  concurrent procedure c;
  var ..
  begin ..
    cobegin
      e;
      f
    coend; ..
  end;
begin ..
  cobegin
    b;
    c;
    d
  coend; ..
end
```

Algoritmo VI.1 - Exemplo de um Programa Concorrente

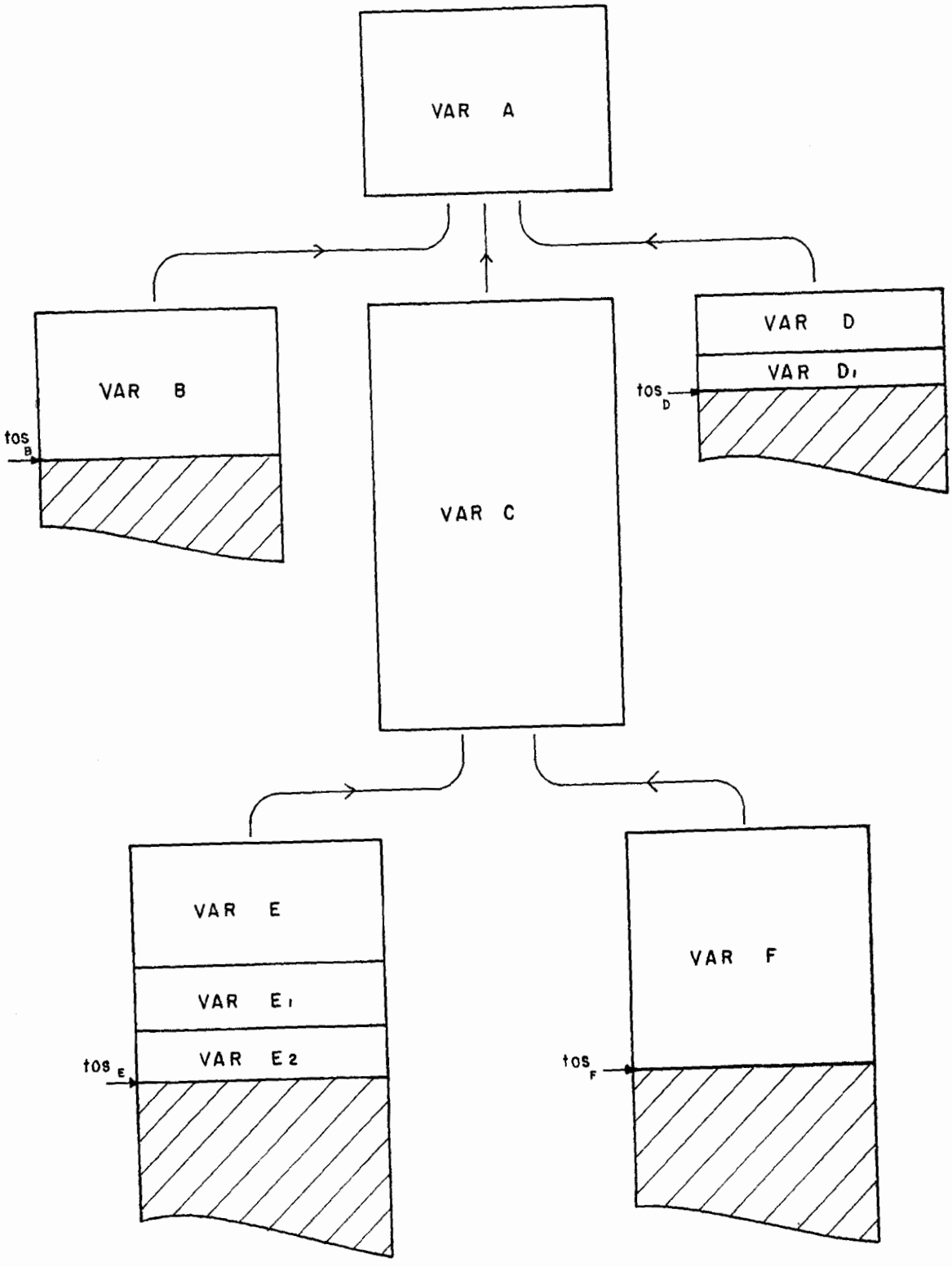


FIGURA VI. 3 - ÁRVORE DE STACK PARA O ALGORITMO VI.1.

Logo que são criados, os ramos de stack alocam inicialmente um registro de ativação para as variáveis declaradas na procedure principal. O espaço adicional é reservado para os outros registros de ativação (como por exemplo E1 e E2 no processo E), que poderão ser construídos durante a execução.

Este arranjo sugere que sejam utilizados dois tipos de descritores para acessar uma determinada variável na estrutura: a identificação do ramo da árvore e a posição relativa da informação dentro do ramo. Este tipo de apontador se adapta bem aos conceitos já estabelecidos, se cada ramo da árvore for associado a um pacote de informação.

Partindo da idéia citada acima, os apontadores passarão a ocupar duas palavras de memória: uma delas indicará o número do pacote, e a outra o deslocamento da informação em relação ao início do pacote. O tipo areaptr descrito abaixo será usado para formalizar as partes que compõe um apontador.

```
type areaptr = record case boolean of
                 true;  (ptr: ^integer);
                 false: (offsetpart: offsetrange;
                         pacpart: pacrange)
               end;
```

Esta alteração não acarretará mudanças radicais na estrutura do sistema UCSD original, pois o compilador, que poderia representar o maior obstáculo, só precisa ser informado do espaço ocupado por uma variável do tipo "pointer", através de uma declaração de constante. As operações que empregam apontadores são bem definidas por códigos intermediários específicos, e dependem quase que unicamente do interpretador que irá implementar o processador virtual.

De acordo com o Capítulo IV, a interligação dos registros de ativação da máquina-P original é realizada através de uma palavra de controle denominada "mark stack" (MSCW). A Figura VI.4 mostra a nova configuração do MSCW, que leva em conta os

novos requisitos de endereçamento virtual. Foram incluídas identificações para os pacotes correspondentes aos links estático (MSPSTAT) e dinâmico (MSPDYN), e os valores de MSDYN e MSSTAT passaram a representar um deslocamento dentro dos seus respectivos pacotes. Foi também incluída a identificação do pacote que contém o registro de ativação global (MSPBASE) nos casos em que MSBASE é utilizado. O campo MSSEG, passa a indicar o número do pacote de código de uma segment procedure e os valores de MSIPC e MSJTAB continuam a representar o ponto de chamada do procedimento em execução, dentro do pacote MSSEG.

Três novos pseudo- registros foram criados na máquina-P estendida, para armazenar as informações relativas aos pacotes que serão mais comumente usados no processamento:

- O primeiro deles é o BSEG, que representa o ramo da árvore de stack que contém o registro de ativação das variáveis globais do programa. O registro BASE, já existente, passa a corresponder a um deslocamento dentro do pacote global;

- Analogamente, foi criado o registro LSEG, para identificar o pacote associado ao "ramo local" da árvore de stack. Os registros MP e SP continuam com as suas funções originais, apontando respectivamente para o registro de ativação local e para o topo do stack, situados dentro deste pacote;

- Por último, é previsto um novo registro para identificar o pacote em que as variáveis dinâmicas estão sendo alocadas (HSEG). O apontador do heap (NP) será substituído por uma estrutura mais complexa, que atenda aos requisitos impostos pela programação concorrente.

Além destas alterações, o registro SEG, que passa a se chamar CSEG, indica agora o número do pacote de código que contém a procedure que está sendo executada. A exemplo do "mark stack", os registros IPC e JTAB correspondem a um deslocamento dentro do pacote de código.

A Figura VI.5 apresenta um mapa contendo exclusivamente os

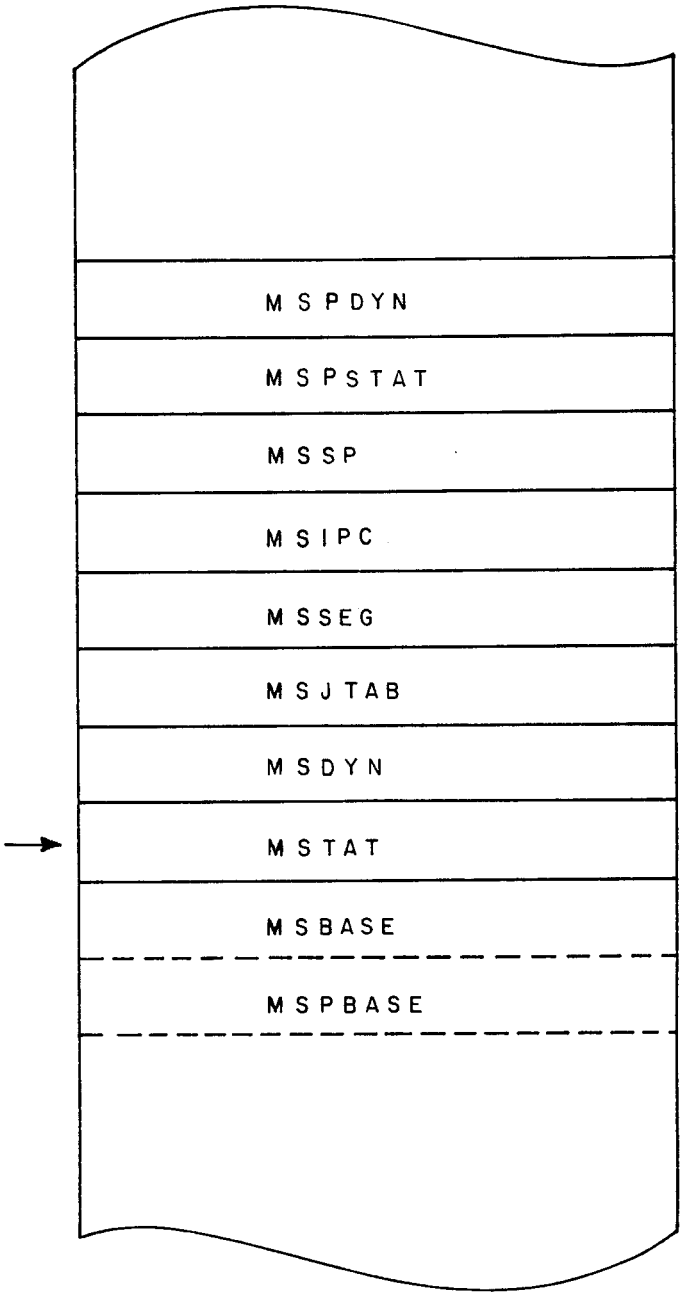


FIGURA VI. 4 - MARK STACK DA MÁQUINA ESTENDIDA.

pacotes envolvidos na execução da procedure E2 do Algoritmo VI.1. Nele estão representadas possíveis posições dos registros da máquina-P estendida, e os "links" estático e dinâmico estabelecidos.

VI.2. GERÊNCIA DE MEMÓRIA

Neste capítulo são descritos os principais procedimentos responsáveis pela manipulação da memória do sistema projetado. Os algoritmos prevêem a distribuição dos espaços virtual e real de endereçamento, levando em conta tanto o sistema-P UCSD original, descrito no Capítulo IV, quanto a arquitetura para multiprocessamento apresentada no Capítulo V. As estruturas de dados utilizadas irão sendo detalhadas à medida que o texto exigir.

A procedure initpac (Algoritmo VI.2) é responsável pela inicialização da tabela de pacotes. O espaço virtual de endereçamento foi igualmente dividido entre os pacotes existentes, através da atribuição de valores sequenciais ao campo base. A Figura VI.6 mostra , através das linhas tracejadas, como são projetados os endereços virtuais na matriz de páginas da memória comum.

Uma lista de páginas livres é implementada com o registro freepag, e o vetor paglink descritos abaixo:

```
type pagrange = 0 .. maxpag
var  freepag : record
                 inic,
                 tam : pagrange
               end;
     paglink : array[pagrange] of pagrange;
```

O campo freepag.inic contém o índice da primeira página livre, e o vetor paglink se encarrega do encadeamento das demais páginas da lista. O campo freepag.tam indica a quantidade de páginas livres existentes na memória física.

```
procedure initpac;
var pac: pacrange;
begin
  for pac := 0 to maxpac do
  with pactbl[pac] do
  begin base := pac;
    tamanho := 0;
    present := false;
    acesso := noaccess;
    tipo := livre;
    bloco := 0;
    wrback := false
  end
end;
```

Algoritmo VI.2 - Inicializacão de Pacotes

```
procedure removearea(nodo: ^dskarea);
begin (* remove nodo da lista freedsk *)
  if nodo^.ant <> nil then nodo^.ant^.post := nodo^.post
    else freedsk.ptr := nodo;
  if nodo^.post <> nil then nodo^.post^.ant := nodo^.ant;
  freedsk.tam := freedsk.tam-nodo^.nblk
end;
procedure insertarea(nodo: ^dskarea);
begin (* insere nodo na lista freedsk, de acordo com o seu tamanho *)
  if (freedsk.ptr = nil) or (nodo.nblk  = freedsk.ptr^.nblk)
  then insere nodo^ no inicio da lista else
  begin
  localiza a posicão em que nodo^ deve ser inserido;
  insere nodo^ na lista
 end;
 freedsk.tam := freedsk.tam + nodo^.blk
end;
```

Algoritmo VI.3 - Operacões na lista de Espaco Livre do Arquivo System.Memory

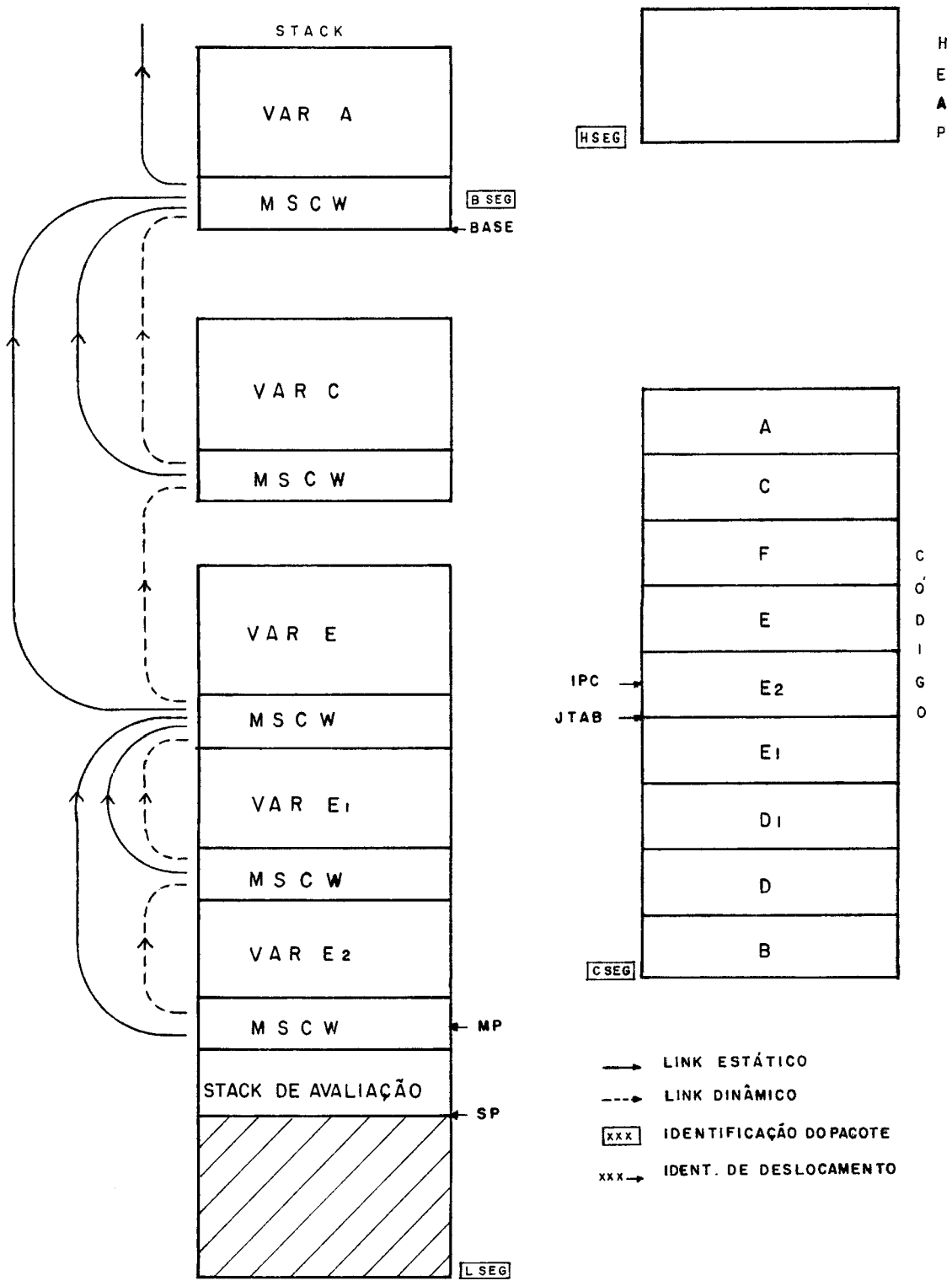


FIGURA VI.5 - MAPA DE MEMÓRIA DA MÁQUINA - P ESTENDIDA PARA O ALGORITMO VI.1 (OS ENDEREÇOS CRESCEM DE BAIXO PARA CIMA).

Um campo da tabela de pacotes chamado inicpag, que ainda não foi descrito, contém o endereço físico da primeira página alocada para um determinado pacote. O encadeamento das demais páginas é novamente feito através do vetor paglink. As páginas da memória física são alocadas de acordo com a quantidade de memória necessária para cada pacote, através do preenchimento de um número adequado de registros na matriz de páginas. A Figura VI.6 mostra um pacote com quatro páginas alocadas, onde a não utilização de todo o espaço virtual a que o mesmo tem direito representa o desperdício de alguns registros na matriz de páginas. Contudo esta alternativa tem a vantagem de dispensar totalmente as operações de compactação na memória, já que o espaço máximo que pode ser alocado para um pacote já está previamente reservado no endereçamento virtual.

Na atual implementação, os pacotes não residentes na memória física estão armazenados num arquivo em disco chamado SYSTEM.MEMORY. As informações contidas na tabela de pacotes (unidade, tamanho e bloco) serão usadas como parâmetros para as primitivas básicas de E/S (unitread e unitwrite), que deverão realizar as operações de acesso ao arquivo. O espaço livre neste arquivo será gerenciado através de uma lista duplamente encadeada mostrada na Figura VI.7 e descrita pelo registro freedsk:

```
type dskarea = record
                 iblk,
                 nblk: blkrange;
                 ant,
                 prox: ^dskarea
               end;
var  freedsk : record
                 ptr: ^dskarea;
                 tam: blkrange
               end;
```

Cada nodo da lista representa um espaço vazio no arquivo SYSTEM.MEMORY, que tem o seu tamanho e início descritos por nblk

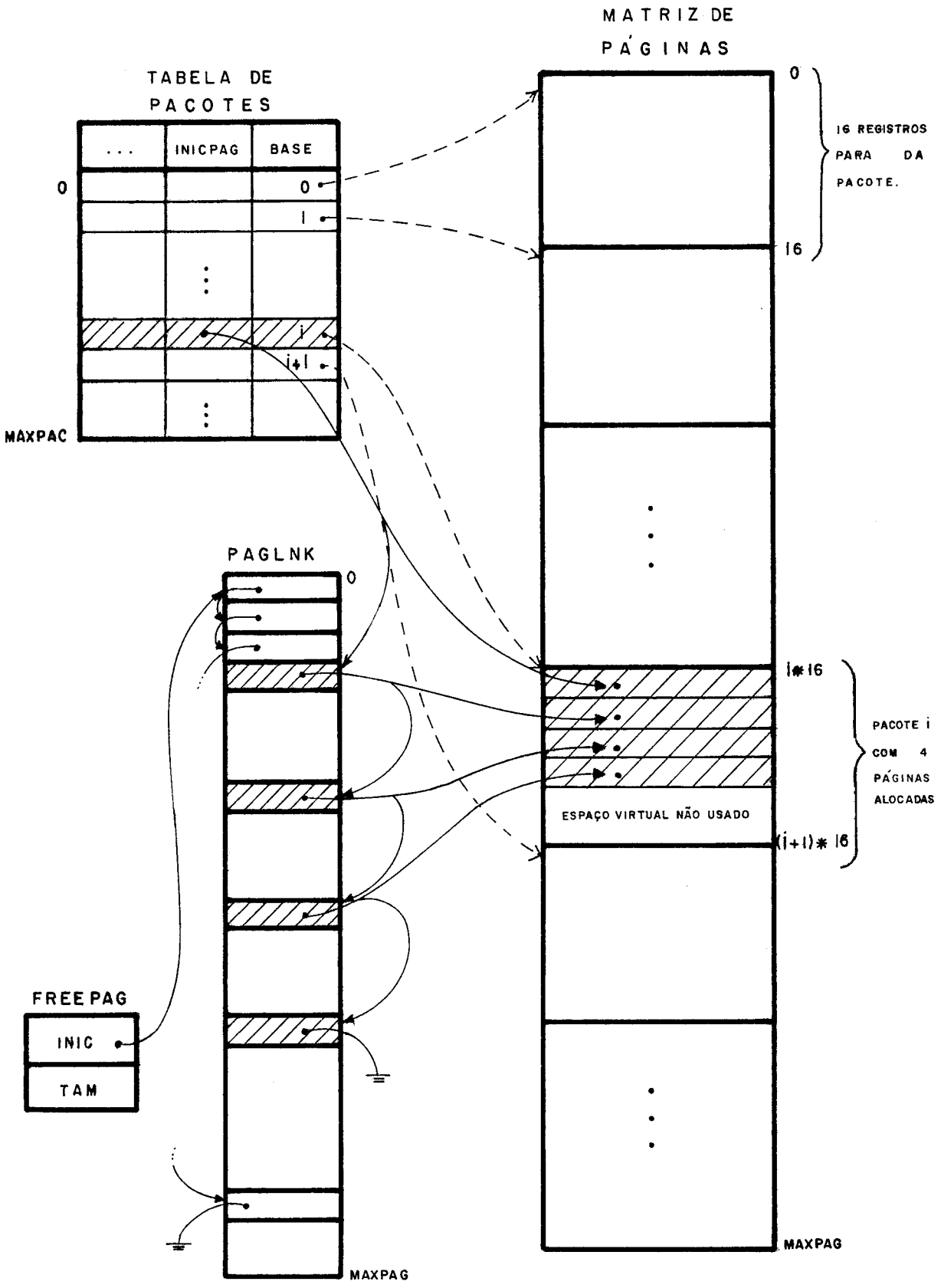


FIGURA VI. 6 - ESTRUTURA DE DADOS PARA ENCADEAMENTO DE PÁGINAS.

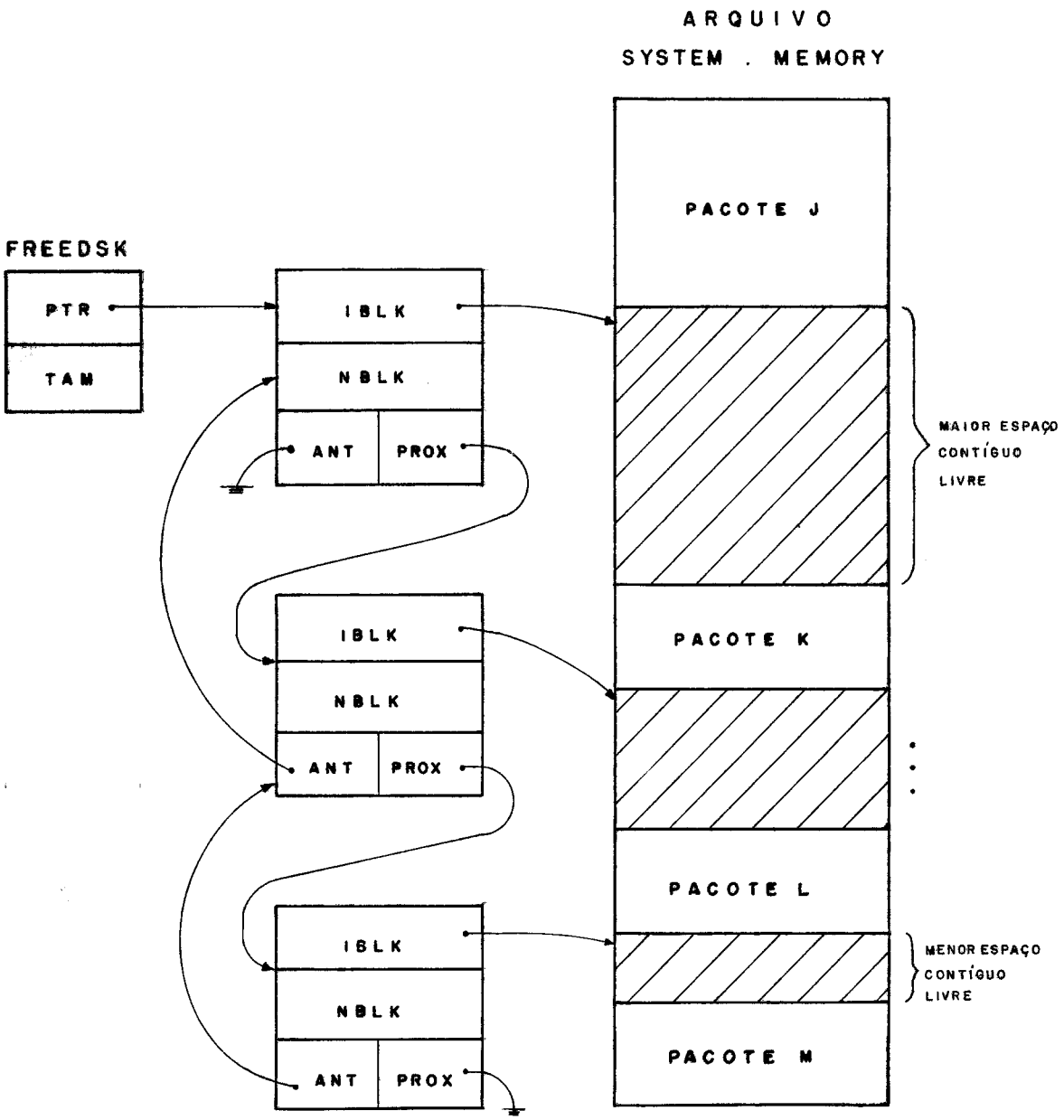


FIGURA VI. 7 - GERENCIAMENTO DO ESPAÇO LIVRE DO ARQUIVO SYSTEM.MEMORY.

e iblk respectivamente. A lista está classificada de acordo com o tamanho associado ao nodo, de tal forma que freedsk.ptr^ representa a maior área contígua disponível no arquivo. Os apontadores ant e post interligam os nodos nos dois sentidos: crescente e decrescente respectivamente. O tamanho do espaço total disponível no arquivo é dado em freedsk.tam. O Algoritmo VI.3 resume as características das duas procedures que deverão manipular os nodos desta lista: removearea e insertarea.

A procedure unloadpac, descrita pelo Algoritmo VI.4 é usada para remover um pacote da memória física. Se o parâmetro keep for verdadeiro, verifica-se a necessidade de atualização ou mesmo da criação de uma cópia do pacote no arquivo SYSTEM.MEMORY. Caso contrário, a área no arquivo associada ao pacote é liberada através da procedure libdskcopy. Finalmente, as páginas alocadas para o pacote são tornadas disponíveis, e o flag de presença do pacote na tabela de páginas assume o valor falso, indicando sua ausência da memória física. A procedure compactadisk realiza a compactação do arquivo SYSTEM.MEMORY, quando não houver espaço contíguo suficiente para conter todo o pacote. A função roundup realiza a divisão dos dois operandos, com aproximação para o inteiro superior mais próximo.

A procedure loadpac realiza a operação inversa da rotina anteriormente descrita, ou seja, carrega na memória física um pacote contido no arquivo SYSTEM.MEMORY. Este procedimento é ativado, na maioria dos casos, após a ocorrência da interrupção de "pacote ausente", gerada pelo processador virtual. São utilizadas as informações de unidade, bloco e tamanho, existentes na tabela de pacotes, para preencher os parâmetros da primitiva unitread. Se não houver espaço suficiente na memória física para armazenar todo o pacote, a procedure releasepag, descrita no Algoritmo VI.6 é chamada. A função pacescolhido, desenvolve uma política de arbitragem, levando em consideração parâmetros tais como a prioridade relativa entre as tarefas, os tipos de pacotes residentes na memória e a estrutura da árvore de stack, com o objetivo de optar entre a remoção de alguns pacotes da memória ou a suspensão da tarefa que está sendo executada.

```
procedure unloadpac(pac: pacrange,keep: boolean):
var areasize: blkrange;
    freeblk: ^dskarea;
    area: areaptr;
    endpag: pagrange;
begin
  with pactbl[pac] do if present then
  begin if keep and wrback then
    begin if bloco = 0 then
      begin areasize := roundup(tamanho,blksize);
        if freedsk.tam < areasize then trap(diskmemovf);
        freeblk := freedsk.ptr;
        if freeblk^.nblk < areasize then compactadsk(areasize);
        while freeblk^.ptr^.nblk > areasize do freeblk := freeblk^.ptr;
        removearea(freeblk);
        freeblk^.nblk := freeblk^.nblk-areasize;
        bloco := freeblk^.iblk+freeblk^.nblk;
        insertarea(freeblk);
      end;
      area.pacpart := pac;
      area.offserpart := 0;
      unitwrite(unidade,area,ptr,tamanho,bloco);
    end else if not keep then libdiskcopy(pac);
    endpag := inicpag;
    while paglnk[endpag] <> nilpag do endpag := paglnk[endpag];
    paglnk[endpag] := freepag.inic;
    freepag.inic := inicpag;
    freepag.tam := freepag.tam+roundup(tamanho,pagsize);
    present := false
  end
end;
```

Algoritmo VI.4 - Remocão de um Pacote da Memoria Fisica

```
procedure loadpac(pac: pacrange);
var npag,g: pagrange;
    area: areaptr;

begin
  npag := roundup(pactbl[pac].tamanho,pagsize);
  if freepag.tam < npag then releasepag(npag)
  for g := 1 to npag do
  begin setpag(pac*pagperpac+g,freepag.inic);
    freepag.inic := paglnk[freepag.inic]
  end;
  area.pacpart :=pac;
  area.offsetpart := 0;
  with pactbl[pac] do
  begin
    unitread(unidade.area,ptr,tamanho,bloco);
    present := true;
    wrback :=false;
    util := 1
  end
end;
```

Algoritmo VI.5 - Carregamento de um Pacote na Memoria Fisica

```
procedure releasepag(npag: pagrange);
var pac: pagrange;
  function pacescolhido(freepagsize: pagrange): pacrange;
  begin
    (* indica o pacote mais adequado para deixar a memória física *)
  end;
begin
  while freepag.tam < npag do unloadpac(pacescolhido,true);
end;
```

Algoritmo VI.6 - Liberacão Forcada de Paginas da Memoria Fisica

As operações realizadas até o presente momento, não levaram em conta o tipo de pacote que está sendo manipulado. Contudo, como a tabela de pacotes será utilizada tanto para representar os trechos de código quanto as áreas de dados, é preciso levar em consideração as diferentes características associadas a cada um destes dois tipos de informação. Os pacotes de código deverão ser alocados sempre que um programa começar a executar, de tal forma que cada segment procedure corresponda a um pacote. Uma restrição imposta pelo processador virtual aos pacotes de código de um programa, é que eles estejam contiguamente alocados na tabela de pacotes. Assim, a cada programa ativo estará associado um valor que irá indicar o pacote inicial, a partir do qual serão acessadas as diversas segment procedures existentes no programa. A Figura VI.8 ilustra um caso típico, em que o programa A é composto por três segment procedures.

Uma característica importante dos pacotes de código é a possibilidade de relocação do seu endereço virtual. Desde que seja respeitada sua contiguidade, é possível haver um deslocamento dos pacotes de código de um programa para qualquer posição da tabela de pacotes, alterando consequentemente os seus endereços virtuais. Obviamente, a referência ao pacote inicial de código do programa tem que ser mantida consistente com a sua posição.

Os pacotes de dado são normalmente alocados individualmente, quer sejam utilizados para representar um espaço estático de armazenamento ou estejam associados aos pacotes para alocação dinâmica de memória. Entretanto, uma vez alocados, estes pacotes não poderão mudar seu endereço virtual, em virtude das referências absolutas eventualmente estabelecidas durante a execução do programa.

As características dos pacotes de código e dados sugerem uma distribuição, onde o espaço existente na tabela de pacotes é dividido em duas regiões: uma delas, reservada aos pacotes de código, poderá ser compactada sempre que necessário, enquanto que a outra, exclusiva para dados só admitirá operações de

```
PROGRAM A

. . .

PROCEDURE A1 ;
BEGIN . . . END;

SEGMENT PROCEDURE A2 ;
BEGIN . . . END;

SEGMENT PROCEDURE A3 ;
BEGIN . . . END;

BEGIN
. . .
END.
```

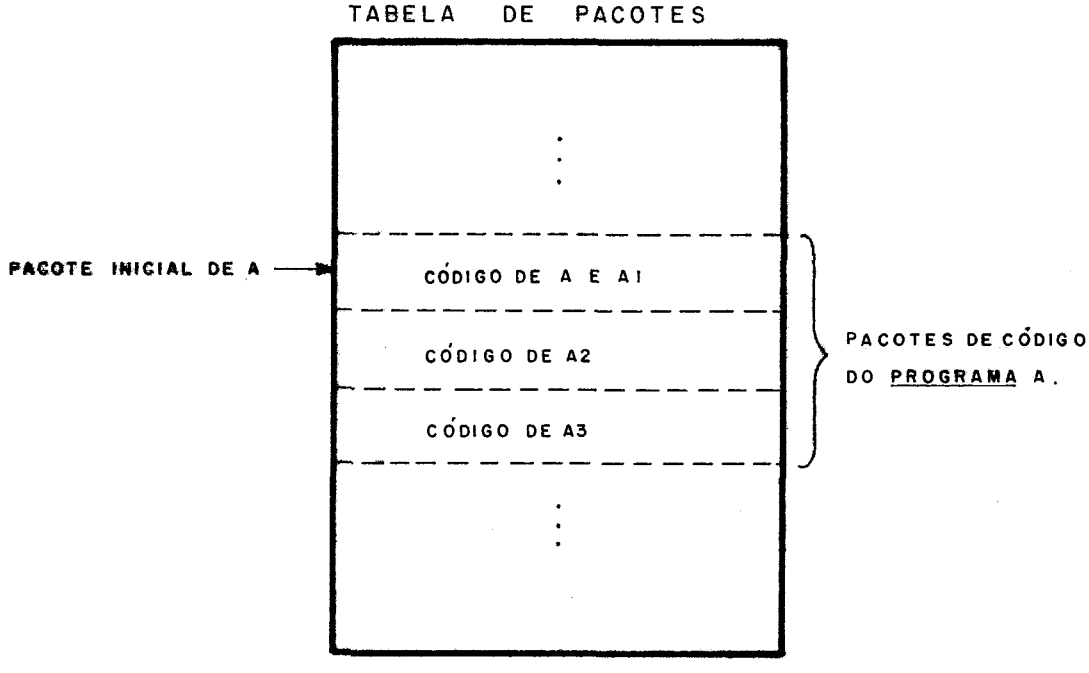


FIGURA VI. 8 - EXEMPLO DE DISTRIBUIÇÃO DOS PACOTES DE CÓDIGO DE UM PROGRAMA.

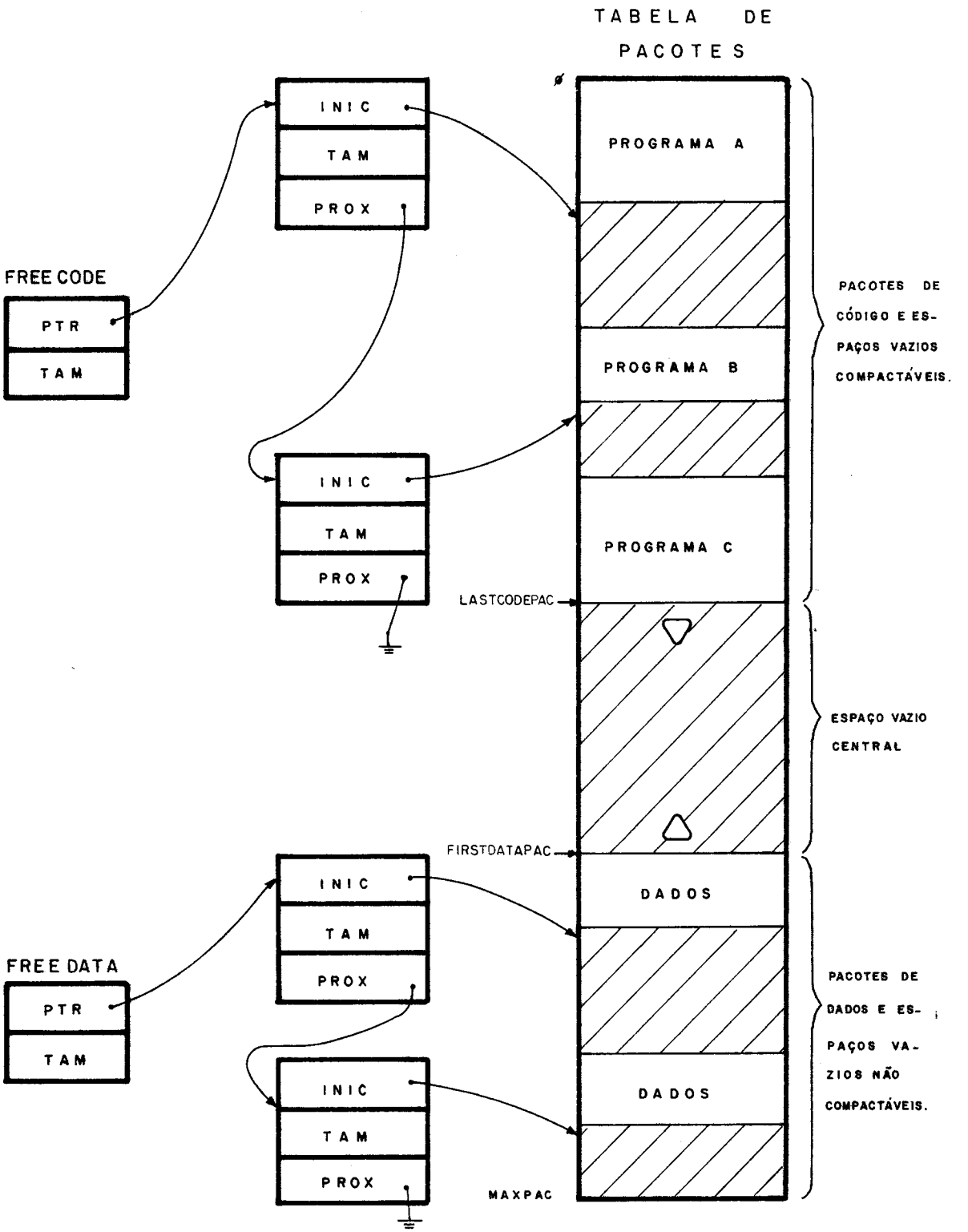


FIGURA VI. 9 - GERÊNCIA DA TABELA DE PACOTES.

alocação e liberação de espaço virtual. A Figura VI.9 mostra a estrutura de dados que é usada para gerenciar a tabela de pacotes. O registro freepac, mostrado a seguir indica o espaço virtual disponível.

```
type paclist = record
                 inic,
                 tam: pacrange;
                 prox: ^paclist
               end;
var  freepac: record
                freecode,
                freedata: record
                            ptr: ^paclist;
                            tam: pacrange
                          end;
                lastcodepac,
                firstadatapac: pacrange
              end;
```

Os valores de lastcodepac e firstdatapac delimitam o espaço vazio central da tabela, que pode ser utilisado tanto para alocar pacotes de código quanto de dados. Os espaços vazios dentro da região de código (acima de lastcodepac) e de dados (abaixo de firstdatapac) são representados através das listas freecode e freedata respectivamente. Estas listas são ordenadas de acordo com o tamanho dos espaços livres, em ordem decrescente.

A procedure alocpac, mostrada no Algoritmo VI.7 é usada para criar um novo espaço na memória virtual. O parâmetro início é devolvido com o número do pacote inicial correspondente ao grupo com npac pacotes. De acordo com ptipo, um grupo contíguo de pacotes é alocado na região de código ou de dados da tabela de pacotes.

A procedure liberapac, realiza a função inversa da anterior, liberando os espaços virtuais que já não são mais necessários. A chamada a esta rotina ocorre normalmente no fim da execução de um programa para liberar os pacotes de código do mesmo, ou então

```
procedure alocpac(var inicio: pacrange; ptipo: pactype,npac: pacrange);
var pt: ^paclist;
    existepac: boolean;
    c: pacrange;
begin with freepac do
  begin existepac := (firstdatapac-lastcodepac-1+freecode.tam)> = npac;
    case ptipo of
      codigo: begin if not existepac then waitforpac(npac);
                pt:= menor nodo da lista freecode c/mais que npac pacotes;
                if pt = nil then
                begin if (firstdatapac-lastcodepac-1) < npac
                then compactacode;
                ajusta inicio no espaco vazio central;
              end else ajusta inicio no nodo pt^ encontrado
      statdat,
      dyndat: begin existepac := existepac or (freedata.ptr^.tam > = npac);
                if not existepac then waitforpac(npac)
                pt:= menor nodo da lista freedata c/mais que npac pacotes;
                if pt = nil then
                begin if (firstdatapac-lastcodepac-1)<npac
                then compactacode;
                  ajusta inicio no espaço vazio central;
                end else ajusta inicio no nodo pt^ encontrado;
              end;
      livre:  trap(invpactype)
    end
 end;
 for c:= inicio to inicio+npac-1 do inicializa campos de pactbl[c]
end;
```

Algoritmo VI.7 - Alocacão de Memoria Virtual

no caso de término de um procedimento concorrente, quando os ramos de stack se fundem. Como veremos mais adiante, algumas funções de liberação de áreas para alocação dinâmica de memória também irão se utilizar deste procedimento. Por medida de segurança, só é permitida a liberação de um grupo homogêneo de pacotes, ou seja, todos devem ter o mesmo tipo. O Algoritmo VI.8 resume os passos necessários à implementação desta função.

O sincronismo entre as tarefas que disputam os recursos de memória é realizado por dois procedimentos: waitforpac e signaltopac. Estas funções são implementadas através de uma fila associada a um semáforo binário. Sempre que não houver pacotes disponíveis, a tarefa é suspensa com a execução de uma chamada a waitforpac. Após cada liberação de espaço virtual uma chamada a signaltopac reativa as tarefas que eventualmente possam estar esperando na fila. Maiores detalhes sobre como implementar estes procedimentos podem ser encontrados nos estudos de Barbosa et al (69).

Existem mais duas funções, exclusivas para os pacotes de dados, que permitem a variação do tamanho do espaço físico a eles associados. A procedure extendepac aumenta o número de páginas de memória de um pacote, enquanto que a procedure reduzpac realiza a operação inversa. Estas facilidades estão representadas no Algoritmo VI.9.

Considerando-se que na atual implementação:
- cada nodo da árvore de stack representa um processo;
- cada processo corresponde a uma procedure do programa e que
- a estruturas da linguagem não permite a interação entre procedures de programas diferentes,

chega-se à conclusão de que toda a subárvore de processos desenvolvida a partir da ativação de um programa só pode ser composta por processos (ou procedures) que façam parte do código original do programa. Não estão incluídos neste grupo os procedimentos intrínsecos da linguagem que são suportados pelo sistema operacional. O efeito prático desta característica pode ser observado na Figura VI.10, onde são representados os

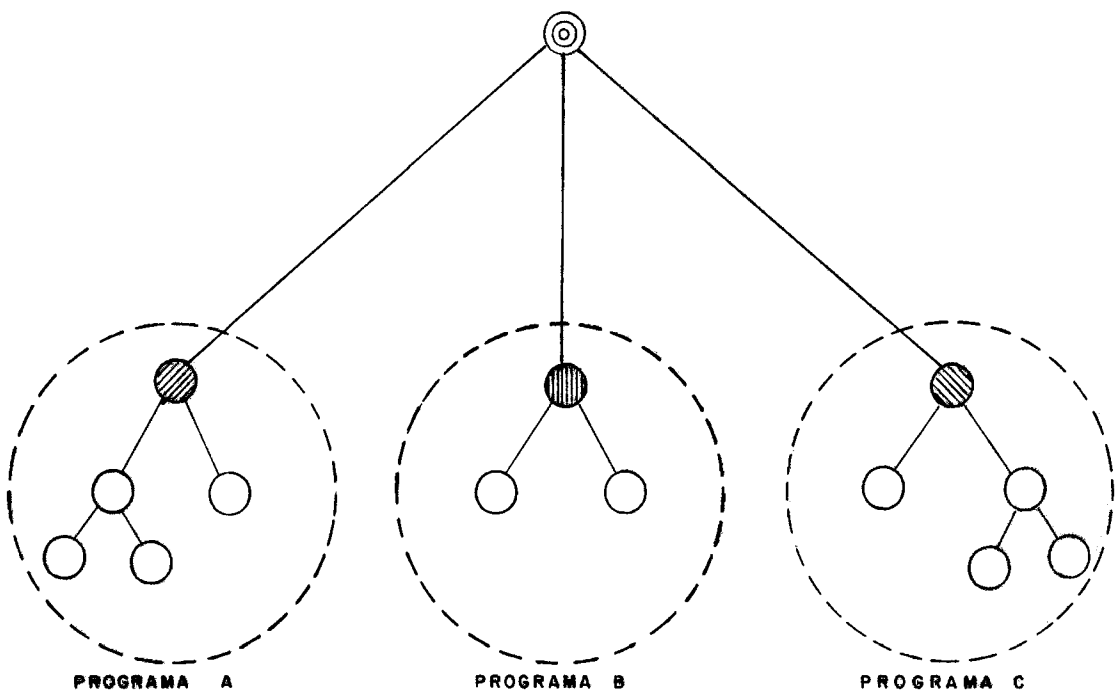
```
procedure liberapac(pac: pacrange,npac: pacrange);
var pt: ^paclist;
    c: pacrange;
begin
  for c := pac+1 to pac+npac-1 do
    if pactbl[c].tipo <> pactbl[pac].tipo then trap(invpacoper);
  for c := pac to pac+npac-1 do unloadpac(c,false);
  with freepac do
  case pactbl[pac].tipo of
    codigo: if pac = lastcodepac-npac+1 then lastcodepac := pac else
            begin new(pt,sysheap);
               pt^.inic := pac;
               pt^.tam := npac;
               insere nodo pt^ na lista freecode;
               freecode.tam := freecode.tam+npac
            end;
    statdat,
    dyndat: if pac=firstdatapac then firstdatapac := firstdatapac+npac else
            begin new(pt,sysheap);
               pt^.inic := pac;
               pt^.tam := npac;
               insere nodo pt^ na lista freedata;
               freedata.tam := freedata.tam+npac
            end;
     livre:   trap(invpacoper)
end;
signaltopac
end;
```

Algoritmo VI.8 - Liberacão de Memoria Virtual

```
procedure extendepac(pac: pacrange,nbytes: offsetrange);
var g,pagaloc,pagext: pagrange;
begin if pactbl[pac].tipo not in [statdat,dyndat] then trap(invpacoper) else
  begin pagaloc := roundput(pactbl[pac].tamanho,pagsize);
    pagext := roundup(pactbl[pac].tamanho+nbytes,pagsize)-pagaloc;
    if pagext > 0 then
    begin if pagaloc+pagext > pagperpac then trap(pagovf) else
      begin if not pactbl[pac].present then loadpac pac ;
        if freepag.tam < pagext then releasepag(pagext-freepag.tam);
        for g := 1 to pagext do inicializa paginas estendidas;
       freepag.tam := freepag.tam-pagext;
      end
    end;
    pactbl pac .tamanho := pactbl[pac].tamanho+nbytes;
    libdiskcopy(pac)
  end
end;


procedure reduzpac(pac: pacrange,nbytes: offsetrange);
var pagaloc,pagreduz: pagrange; begin
  if pactbl[pac].tipo not in [statdat,dyndat] then trap(invpacoper) else
  begin pagaloc := roundup(pactbl[pac].tamanho,pagsize);
    pagreduz := pagaloc-roundup(pactbl[pac].tamanho-nbytes,pagsize);
    if pagreduz > 0 then
    begin if pagaloc-pagreduz < 0 then trap(pagundflw) else
      begin if not pactbl[pac].present then loadpac(pac);
        transfere paginas em excesso para freepag
      end
    end;
    pactbl[pac].tamanho := pactbl[pac].tamanho-nbytes;
    libdiskcopy(pac)
  end
end;
```

Algoritmo VI.9 - Extensão e Redução do Tamanho dos Pacotes de Dados



REGISTRO DE ATIVAÇÃO DO SISTEMA OPERACIONAL, QUE CONTEM AS VARIÁVEIS COMUNS A TODOS OS PROGRAMAS.

REGISTRO DE ATIVAÇÃO DO PROCESSO INICIAL DE UM PROGRAMA, ONDE SÃO ALOCADAS AS VARIÁVEIS GLOBAIS DO MESMO.

REGISTRO DE ATIVAÇÃO DE UM PROCESSO CONCORRENTE (PROCEDURE) DO PROGRAMA.

FIGURA VI.10 - EXECUÇÃO DE PROGRAMAS CONCORRENTES.

processos de três programas concorrentes.

Em virtude da natureza dinâmica do sistema, os trechos de código do programa, ou mais objetivamente as suas segment procedures, devem ser inscritas para execução no instante em que o programa é ativado, através da procedure alocpac. As chamadas a procedures, deste ponto em adiante, terão sempre como referência o início dos pacotes de código do programa em questão.

Adicionando-se alguns campos ao registro apresentado em Barbosa et al (69). tem-se a estrutura final que descreve os processos da máquina-P estendida:

```
type nodo = record
              pai,prox: ^nodo:
              ntasks: integer;
              filhos: ^nodo;
              prioridade,schedprior: priorange;
              iorslt: integer;
              status: record
                        ipc,jtab,base,mp,sp: offsetrange;
                        cseg,lseg,bseg: pacrange
                      end;
              prog: ^proginfo
            end;
```

- pai aponta para o descritor do processo que criou o processo corrente;
- prox é usado para encadear os processos nas diversas filas existentes no sistema (semáforos, ready, running, E/S, etc.);
- ntasks representa o número de processos-filhos do processo corrente que ainda não terminaram. O processo corrente só pode continuar a sua execução quando ntasks for nulo;
- filhos é uma fila onde irão ser encadeados os processos criados pelo processo corrente (durante um cobegin/coend), até que todos possam ser simultaneamente ativados;
- prioridade serve para posicionar um processo dentro de uma fila genérica, enquanto que schedprior é usado exclusivamente para determinar o escalonamento dos processos encadeados na

fila ready;

- iorslt é usado para armazenar a palavra de estado correspondente à última operação de e/s realizada pelo processo;
- status é um registro que irá conter o estado da máquina estendida, sempre que for necessário suspender o processo por algum motivo e
- prog aponta para o descritor do programa que está associado ao processo corrente. Este campo será detalhado a seguir.

O processo que estiver dependendo de alguma condição para continuar a sua execução (operação em semáforo, e/s, etc.) estará encadeado na fila correspondente a esta mesma condição (fila do semáforo, do monitor de e/s, etc.). Caso contrário, o processo poderá estar ou na fila de processos prontos para executar (ready) ou na fila de processos em execução de um processador (running).

Conforme foi mencionado anteriormente, cada processo concorrente está associado a um determinado programa. Para descrever um programa será utilizado o registro proginfo, apresentado a seguir:

```
type proginfo = record
                    initcodepac,
                    ncodepac: pacrange;
                    nome: tid;
                    prioridade: priorange;
                    vmemunit: unitnum;
                    link: ^proginfo;
                    heap: record
                              mestre,
                              atual,
                              link: ^heaplist
                          end
                end;
```

- initcodepac e ncodepac descrevem o bloco de pacotes correspondentes ao grupo de segment procedures do programa. A

partir destes valores, o processador virtual pode localizar na tabela de pacotes o endereço do código de uma segment procedure.

- nome contém a identificação do programa, sendo normalmente preenchido com o título do arquivo de código que o gerou.
- prioridade contém o valor "default" a ser utilizado em todos os processos criados durante a execução do programa.
- vmemunit indica, para o gerente de memória, qual a unidade de disco que deverá ser usada para implementar a memória virtual do programa. Na atual implementação existirá apenas uma unidade com esta finalidade (SYSUNIT).
- link serve para interligar os diversos programas criados no sistema.
- o heap é um registro que está associado à gerência do espaço virtual usado para alocação dinâmica de memória ("heap"). Maiores informações sobre o "heap" serão fornecidas a seguir.

As funções previstas na máquina sequencial para alocação dinâmica de memória eram implementadas através de uma única pilha, localizada em um dos extremos da memória, conforme mostra o Capítulo IV. Esta estrutura não se adapta aos requisitos da máquina expandida, que requer "heaps" diferentes para os diversos programas concorrentes.

Cada programa terá acesso a um ou mais pacotes de "heap", sobre os quais serão permitidas as operações new e dispose previstas por Wirth. O "heap" inicial do programa será estabelecido automaticamente após a sua ativação, havendo ainda a possibilidade de criação de "heaps" adicionais através de construtos especiais da linguagem.

A facilidade de criação e destruição de diversos "heaps" por programa substitui (funcionalmente) a função release prevista na máquina sequencial, e bastante útil em certos casos.

O conjunto de "heaps" acessíveis a um programa é representado por um campo de proginfo que contém três apontadores:

- mestre aponta para o "heap" que estava em vigor antes do

programa ser ativado. Nesta região são armazenadas algumas variáveis que controlam a execução do programa, como por exemplo a própria lista de "heaps".

- atual indica o "heap default" que está sendo usado pelo interpretador para alocar/liberar as variáveis dinâmicas do programa.
- link aponta para o início da lista de "heaps" já criados para o programa.

Cada "heap" é representado por uma estrutura que engloba os seguintes campos:

```
type heaplist = record
                    pac: pacrange;
                    uso: integer;
                    free: array 0..15 of offsetrange;
                    ant,
                    prox: ^heaplist
                end;
```

- pac é o número do pacote que contém o "heap";
- uso é empregado para impedir que um "heap" seja liberado em um momento impróprio, quando por exemplo outros programas estiverem utilizando-o como "mestre".
- free contém a estrutura que irá definir o espaço livre existente no "heap".
- ant e prox são apontadores da cadeia de "heaps" do programa.

A Figura VI.11 exemplifica uma distribuição em que o programa A, após ter criado seu segundo "heap", ativou o programa B. O programa B, por sua vez, teve seu "heap" inicial criado durante a fase de ativação.

A função novoheap" mostrada no Algoritmo VI.10, é usada para criar um novo pacote de "heap" para um determinado programa. Um apontador para o nodo criado é fornecido, para que o mesmo possa ser posteriormente utilizado. O heap inicial ocupa apenas uma página física de memória, mas pode ser estendido à medida que for necessário mais espaço.

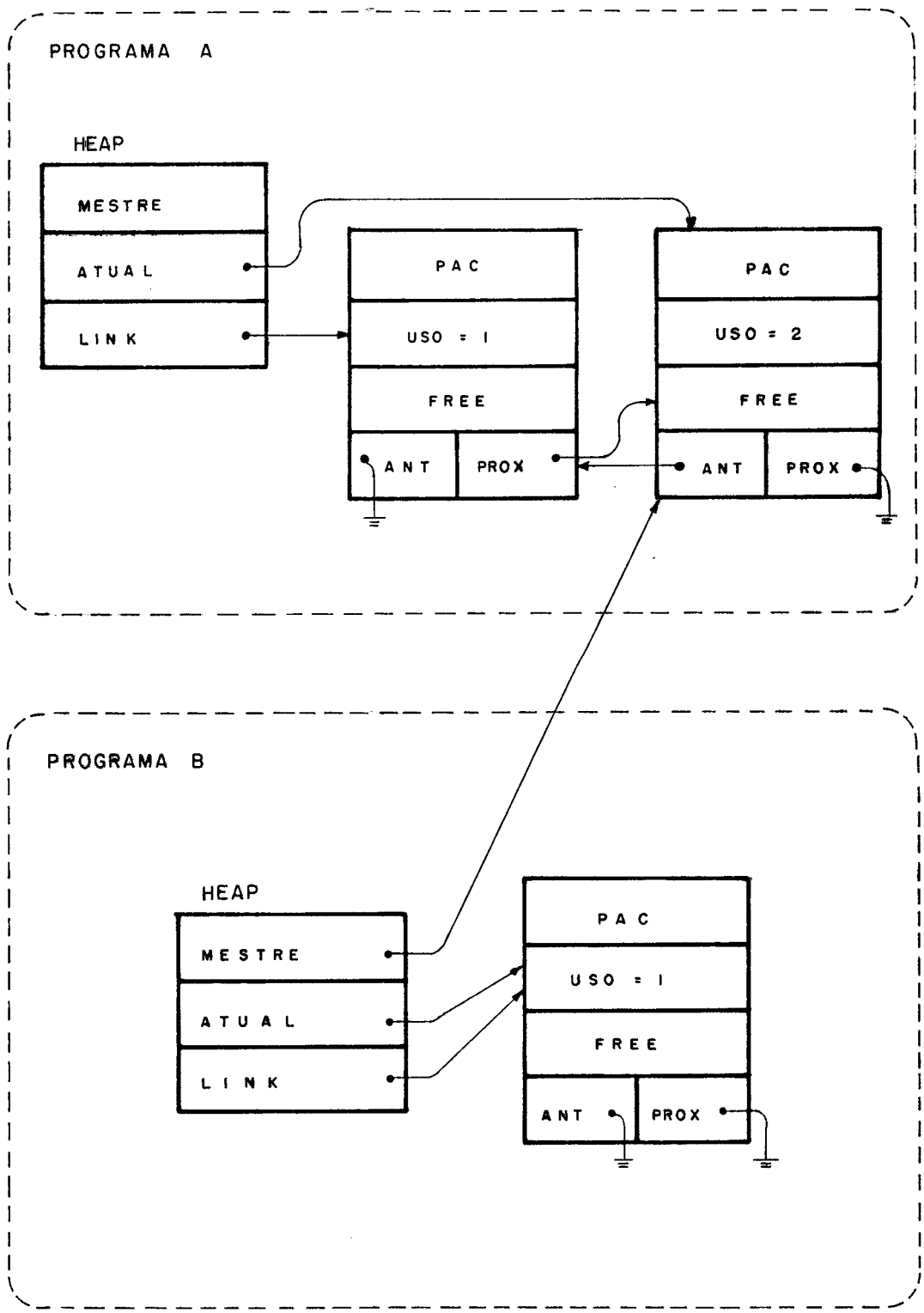


FIGURA VI.11 - EXEMPLO DE UMA DISTRIBUIÇÃO DE "HEAPS".

```
function novoheap: ^heaplist;
var newpac: pacrange;
      h: ^heaplist
begin
  alocpac(newpac,dyndat,1);
  pactbl[newpac].unidade := running^.prog^.vmemunit;
  new(h,running^.prog^.heap.mestre);
  h^.pac := newpac;
  h^.uso := 1;
  h^.ant := nil;
  h^.prox := running^.prog^.heap.link;
  with running^.prog^.heap do
  begin
    if link <> nil then link^.ant := h;
    link := h;
    atual := h
  end;
  extendepac(newpac,pagsize);
  inicializa espaço livre do heap;
  novoheap := h
end;


procedure setheap(newheap: ^heaplist);
var h: ^heaplist; begin
  h := running^.prog^.heap.link;
  while (h <> nil) or (h <> newheap) do h := h^.prox;
  if h = nil then trap(invheapoper) else
    running^.prog^.heap.atual := newheap
end;
```

Algoritmo VI.10 - Criacão e Selecão das Estruturas de Heap

```
procedure releaseheap(oldheap: ^heaplist);
var h: ^heaplist;
begin
  h := running^.prog^.heap.link;
  while (h <> nil) or (h <> oldheap) do h := h^.prox;
  if h = nil then trap(invheapoper) else
  with oldheap^ do
  if uso > 1 then uso := uso-1 else
  begin
    if ant <> nil then ant^.prox := prox;
    if prox <> nil then prox^.ant := ant;
    liberapac(pac,1);
    with running^.prog^.heap do if atual = oldheap then atual:= link;
    dispose(oldheap)
  end
end;
```

Algoritmo VI.11 - Liberacão de Heaps

```
procedure new(var ptr: areaptr; heap: ^heaplist);
var a: offsetrange;
begin
 if heap = nil then trap(noheapavail) else
 with heap^do
 begin
   a = espaço vazio com tamanho sizeof(ptr^), retirado de heap^.free;
   ptr.offsetpart := a;
   ptr.pacpart := pac
 end
end;
procedure dispose(ptr: areaptr,heap: ^heaplist);
begin
 if ptr.pacpart <> heap^.pac then trap(invheapoper);
 libera espaço ocupado por ptr^, inserindo-o em heap^.free
end;
```

Algoritmo VI.12 - Operacões New e Dispose

A procedure setheap, também apresentada no mesmo algoritmo, tem a função de alterar o "heap default" do programa. Só os "heaps" catalogados na lista do programa podem ser usados como "default".

A procedure releaseheap, do Algoritmo VI.11, foi criada para permitir que uma área dinamicamente alocada em diversos passos possa ser liberada de uma só vez. Novamente, um programa só tem o direito de liberar os "heaps" que lhe pertencem. Se o "heap" estiver sendo usado por outros programas (para armazenar variáveis de controle) o valor de uso será decrementado, e a sua liberação só será efetivamente realizada quando o último usuário executar um releaseheap.

As operações new e dispose da máquina estendida consideram, ao manipular com apontadores, a sua composição básica, descrita em areaptr. O Algoritmo VI.12 apresenta, sem entrar em considerações sobre o gerenciamento do espaço livre, a sequência de processamento utilizada.

VI.3 ATIVAÇÃO DINÂMICA DE PROGRAMAS

O exemplo citado a seguir utiliza as ferramentas e recursos já mencionados anteriormente, com o objetivo de implementar um procedimento para ativação dinâmica de programas. Este procedimento pode ser aplicado tanto na construção de um supervisor para um sistema operacional multi-usuários quanto na especificação de um sistema dedicado que necessite de multiprogramação.

Alguns mecanismos foram baseadas na máquina-p sequencial,para que fosse mantida a compatibilidade com a estrutura do sistema original.

O Algoritmo VI.13 apresenta uma descrição do processo runprog, que foi dividido de acordo com as características funcionais dos passos a serem realizados.

```
process runprog(progname: tid, progprior: priorange);
var actprog: ^proginfo;
    codefib: fib;
    segtbl: segtblrec;
    lsegcodeinicial: pacrange;
    h: ^heaplist;
begin setpriority(progprior);
  fopen(codefib,prog,...);
  unitread(codefib.funit,segtbl,sizeof(segtbl), codefib.fheader.firstblk);
  alocpac(codeinicial, codigo,segtbl.lastseg+1);
  for lseg := 0 to segtbl.lastseg do
  with pactbl[lseg+codeinicial],segtbl.diskinfo[lseg], codefib do
  begin unidade := funit;
    bloco := diskaddr+fheader.dfirstblk;
    tamanho := codeleng;
  end;
  new(actprog);
  with actprog^ do
  begin initcodepac := codeinicial;
    ncodepac := segtbl.lastseg+1;
    nome := progname;
    prioridade := progprior;
    vmemunit := sysunit;
    link := running^.prog;
    heap.mestre := running^.prog^.heap.atual;
    heap.atual := nil;
    heap.link := nil
  end;
  with running^.prog^.heap.atual do uso := uso+1;
  running^.prog := actprog;
  h := novoheap;
  userprogram;
  fclose(codefib,...)
  liberapac(actprog^.initcodepac, codigo,actprog.ncodepac);
  h := actprog^.heap.link;
  while h <> nil do begin releaseheap(h); h := h^.prox end;
  running^.prog := actprog^.link;
  h := actprog^.heap.mestre;
  dispose(actprog);
  releaseheap(h);
  setpriority(running.prog^.prioridade)
end;
```

Algoritmo VI.13 - Ativacão Dinamica de Programas

Inicialmente, a prioridade do processo é ajustada de acordo com o parâmetro progprior, pois considera-se que o processamento, a partir da ativação do programa, será realizado segundo as características do mesmo.

A seguir é chamada a procedure initcode, que baseada no supervisor original do Sistema-p, providencia a alocação do espaço virtual de código para o programa.

O registro actprog, contendo informações sobre o programa é criado no heap do programa que está sendo executado, e inicializado através da procedure initprogram. São preenchidos os seus diversos campos, incluindo heap.mestre, que é ajustado para o valor de heap.atual do programa corrente. Incrementa-se heap.atual.uso para garantir a integridade do heap.mestre até o fim da execução do programa. A procedure novoheap cria uma área inicial para alocação dinâmica de memória do programa.

A procedure useprogram, que representa o corpo principal do programa, está associada ao primeiro segmento existente no arquivo de código. A sua chamada corresponde portanto à execução propriamente dita do programa que está sendo ativado. Eventualmente o código do programa poderá incluir a ativação a outros programas, fazendo com que estes processos se repitam outras vezes.

No fim da execução da procedure userprogram, o controle é retornado ao processo runprog novamente, que é responsável pelo restabelecimento do programa que estava sendo executado. Os pacotes de código e do heap são liberados, o processo corrente muda novamente de programa e a prioridade de execução volta a assumir o antigo valor. A Figura VI.12 apresenta uma configuração típica da árvore de stack e da tabela de pacotes nas diversas fases de execução de um programa.

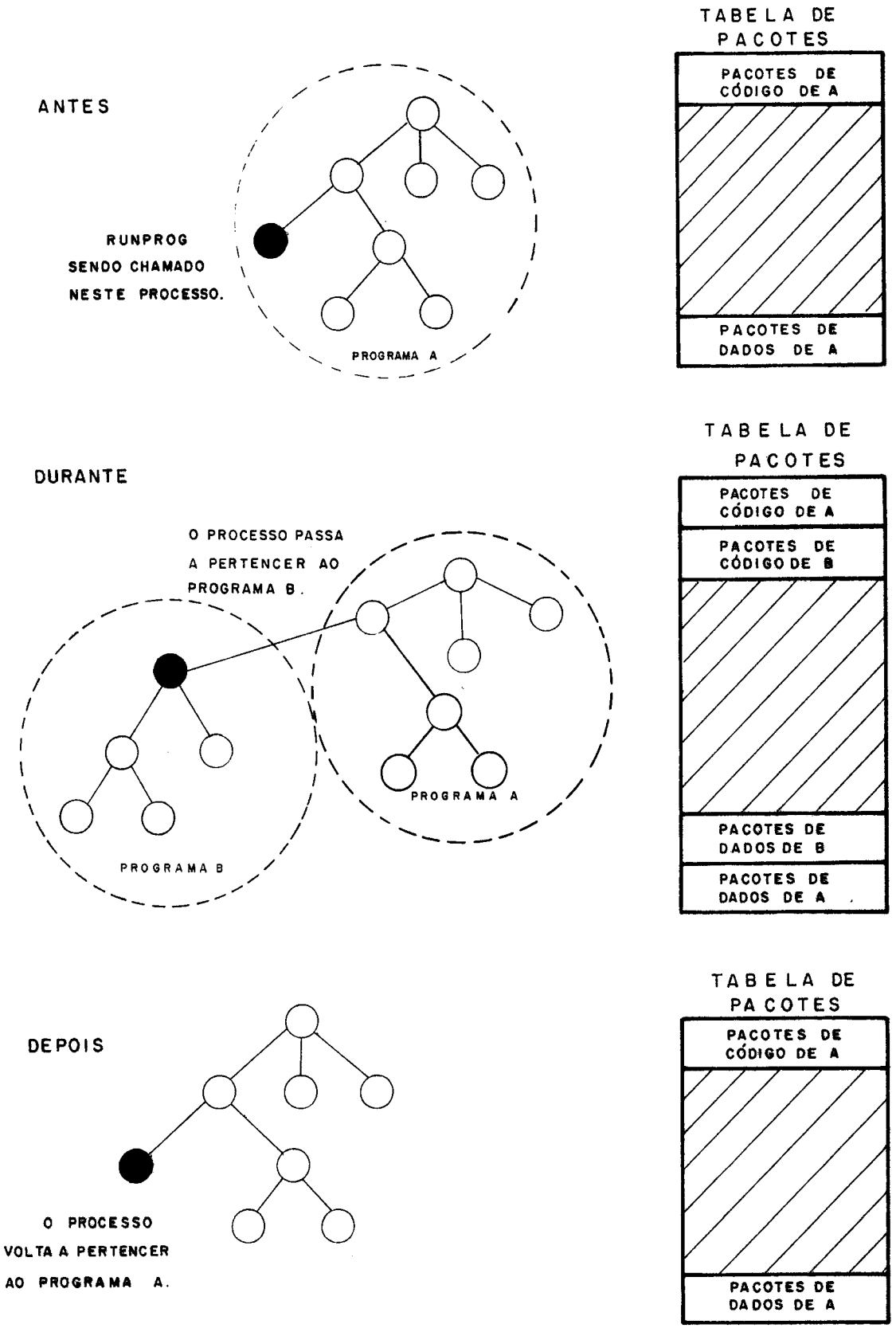


FIGURA VI.12 - FASES DE EXECUÇÃO DO PROGRAMA B.

VII. CONCLUSÃO

A demanda de máquinas mais poderosas nas áreas ligadas à computação tem incentivado investigações nas áreas ligadas à distribuição de processamento. Arquiteturas e componentes adequados ao processamento distribuído tem atualmente apresentado soluções satisfatórias, com relações performance/custo bastante promissoras. Na área de multiprocessamento, em particular, dificuldades de desenvolvimento de software básico tem justificado a inexistência de uma quantidade significativa de sistemas comerciais deste tipo. O desenvolvimento dos trabalhos realizados até o presente momento permite diagnosticar uma espectativa favorável à realização de arquiteturas poderosas utilizando microprocessadores. Apesar da opção inicial pelo microprocessador 8086, há no presente estágio de andamento do projeto, clara tendência de utilização do seu sucessor APX286 como emulador da máquina-p estendida. Outros componentes similares, que vêm sendo anunciados nas revistas técnicas, também parecem preencher de forma mais eficiente os requisitos desejados pelo projeto.

Uma característica adicional, conseguida com o uso da máquina-p, foi a possibilidade de adaptação de uma forma mais amena do sistema às constantes inovações que vêm sofrendo os componentes de hardware da atualidade. Consegue-se assim, garantir um mínimo de continuidade aos projetos de desenvolvimento de software, sem que haja perda de um grande volume de trabalho em razão da obsolescência dos componentes que os implementam.

Finalmente, outros melhoramentos deverão ser feitos à medida que forem obtidos alguns resultados práticos com a implementação da versão ora proposta.

BIBLIOGRAFIA

1 - CEPEL; "Desenvolvimento de um Processador de Funções Avançadas, Baseado em Micro-Processadores, para Treinamento", Proposta de Projeto, no. 185/80, Projeto 7167, Nov 1980.

2 - S. Weber (ed.); "Looking ahead to the year 2000: Techonology", Electronics, vol.53, no.9, pp.530-563, Abr 1980.

3 - J. G. Posa, R. Beresford (eds.); "Techonology update - Semiconductors", Electronics, vol.54, no.21, pp.116-123, Out 1981.

4 - J. G. Posa (ed.); "C-MOS Inspires the best chips yet for Computer, Consumer and Communications Applications", Electronics, vol.54, no.20, pp.103-105, Out 1981.

5 - T. L. Sterling; "Parallel Computer Processing for Power Electronic Networks Simulation", M.Sc./E.E. Thesis, MIT, Cambridge, 1981.

6 - H. Raphael; "Evaluating a Microcomputer Input/Output Performance", Electronics, vol.49, no.17, pp.105-109, Ago 1976.

7 - S. W. Director (ed.); "Special Issue on Computer Aided-Design", Proc. IEEE, vol.69, no.10, pp.1187-1364, Out 1981.

8 - A. W. Burks, H. H. Goldstine, J. von Neumann; "Preliminary Discussion of the Logical Design of an Electric Computing Instrument", Collected works of John von Neumann, The MacMillan Company, vol.5, pp.34-79, 1963.

9 - M. J. Flynn; "Some Computer Organizations and their

Effectiveness", IEEE Trans. on Computers, vol.C-21, no.9, pp.948-960, Set 1972.

10 - L. S. Haynes, R. L. Lau, D. P. Siewiorek, D. Mizell; "A Survey of Highly Parallel Computing", Computer, vol.15, no.1, pp.9-24, Jan 1982.

11 - A. Gottlieb, Schwartz; "Networks and Algorithms for Very-Large-Scale Parallel Computation", Computer, vol.15, no.1, pp.27-36, Jan 1982.

12 - H. T. Kung ; "Why Systolic Architectures?", Computer, vol. 15, no. 1, pp.37-46, Jan 1982.

13 - L. Snyder; "Introduction to the Configurable, Highly Parallel Computer", Computer, vol. 15, pp.47-56, Jan 1982.

14 - H.M. Ahmed, J.M. Delosme; M. Murf; "Highly Concurrent Computing Structures for Matrix Arthmetic and Signal Processing", Computer, vol. 15, no. 1, pp.65-80, Jan 1982.

15 - D.G. Fairbairn; "VLSI: A New Frontier for Systems Designers", Computer, vol. 15, no. 1, pp.87-96, Jan 1982.

16 - R.C. Holt; "Structure of Computer Programs: a Survey", Proceedings of IEEE, vol. 63, no. 6, pp.879, Jun 1975.

17 - H.T. Kung; C.E. Leiserson; "Systolic Arrays (for VLSI)", Sparse Matrix Proc. 1978, Society for Industrial and Applied Mathematics, pp. 256-282, 1979.

18 - A. Mukhopadhyay "Hardware Algorithms for Nonnumeric Computation", IEEE Transaction on Computers, vol. c-28, no. 6, pp.384-394 Jun 1979.

19 - D.A. Huffman; "The Synthesis of Linear Sequential Coding Networks", Information Theory, C. Cherry (ed.), Academic Press, pp.77-95, 1957.

20 - H.T. Kung; "Let's Design Algorithms for VLSI Systems", Proc. Conf. VLSI: Architecture, Design, Fabrication, CALTECH, pp.65-90, Jan 1979.

21 - H.T. Kung; "Special Purpose Devices for Signal and Image Processing: An Opportunity in VLSI", Proc. SPIE. Real-time Signal Processing III, Society of Photo-Optical Instrumentation Engineers, vol.241, pp.76-84, Jul 1980.

22 - H.T. Kung; "Use of VLSI in Algebraic Computation: Some Suggestions", Proc. ACM Symp., Symbolic and Algebraic Computation, ACM Sigsam, pp.218-222, Ago 1981.

23 - J.E. Rodriguez, "A Graph Model for Parallel Computation", MIT Technical Report TR-64, Laboratory for Computer Science, MIT, Cambridge, Mass., Set 1969.

24 - D.A. Adams, "A Computation Model with Data Flow Sequencing", Technical Report CS117, School of Humanities and Science, Stanford University, Stanford, Calif., Dez 1968.

25 - R.R. Seeber; A.B. Lindquist, "Associative Logic for Highly Parallel Systems", AFIPS Conf. Proc., pp.489-493, 1963

26 - T. Agerwala e Arvind (eds.), "Data Flow Systems", Computer, vol. 15, no. 2, pp.10-13, Fev. 1982.

27 - D.D. Gajski; D.A. Padua; D.J. Kuck; e R.H. Kuhn, "A Second Opinion on Data Flow Machines and Languages", Computer, vol. 15, no. 2, pp.58-69, Fev. 1982.

28 - J.B. Dennis, "Data Flow Supercomputers", Computer, vol.13, no. 11, pp.48-56, Nov. 1980.

29 - A.L. Davis e R.M. Keller, "Data Flow Programa Graphs", Computer, vol. 15, no. 2, pp.26-41, Fev. 1982.

30 - W.B. Ackerman, "Data Flow Languages", Computer, vol. 15,

no. 2, pp.15-25, Fev. 1982.

31 - K.J. Thurner, "Parallel Processor Architectures - part 1: General Purpose Systems", Computer Design, vol. 18, no. 1, pp.89-97, Jan 1979.

32 - W.J. Karplus e D. Cohen, "Architectural and Software Issues in the Design and Application of Peripheral Array Processors", Computer, vol. 14, no. 9, pp.11-17, Set 1981.

33 - A. Slade e H.O. McMahon, "A Cryotron Catalog Memory System", Proc. 1956 Eastern Jt. Computer Conf., AIEE, pp.115-120, 1957.

34 - J.A. Rudolph, L.C. Fulmer e W.C. Meilander, "The Coming of Age of the Associative Processor", Electronics, vol. 44, pp.91-96, Fev. 1971.

35 - B.A. Crane, et al, "PEPE Computer Architecture", IEEE Compcon, 1972.

36 - K.E. Batcher, "Flexible Parallel Processing and STARAN", Wescon, 1972.

37 - N. Minsky, "Rotating Storage Devices as Partially Associative Memories", 1972 Fall Joint Computer Conf, AFIPS Conf Proc. Montvale, NJ: AFIPS Press, pp.587-595, 1972.

38 - D.L. Slotnick, "Logic per Track Devices", in Advances in Components, vol. 10 NY: Academic Press, pp.291-296, 1970.

39 - B. Parhami, "Associative Memories and Processors: An Overview and Selected Bibliography", Proc. of the IEEE, vol. 61, no. 6, pp. 722-730, Jun 1973.

40 - C.R. DeFiore e P.B. Berra, "A Quantitative Analysis of the Utilization of Associative Memories in Data Management ", IEEE Trans. on Computers, vol. c-23, no. 2, pp.121-133, Fev. 1974.

41 - R.M. Lea, "Information Processing with an Associative Parallel Processor", Computer, vol. 8, no. 11, pp.25-32, Nov. 1975.

42 - K.J. Thurber e R.O. Berg, "Applications of Associative Processors", Computer Design, vol. 10, no. 11, pp.103-110, Nov. 1971.

43 - T.Lonie, "Array processors: A Selected Bibliography", Computer, vol. 14, no. 9, pp.53-57, Set 1981.

44 - S.P. Hufnagel, "Comparison of Selected Array Processor Architectures", Computer Design, vol. 18, no. 3, pp.151-158, Mar 1979.

45 - W.R. Wittmayer, "Array Processor Provides High Throughput Rates", Computer Design, vol. 17, no. 3, pp.93-100, Mar 1978.

46 - N. Maron, T.A. Brengle; "Integrating an Array Processor into a Scientific Computing Systems", Computer, vol. 14, no. 9, pp.41-44, Set 1981.

47 - J. Allen, "Computer Architecture for Signal Processing", Proceedings of the IEEE, vol. 63, no. 4, pp. 624-633, Abr 1975.

48 - P. Heller, R. Childs e J. Slagev, "Memory Protection Moves onto 16-bit Microprocessor Chip", Electronics, vol. 55, no. 4, pp.133-137, Fev 1982.

49 - S.H.Fuller, J.K.Ousterhout, L.Raskin, P.I.Rubinfield, P.J.Sindhn, R.J.Swan, "Multi-microprocessors: An Overview and Working Example", Proc. IEEE, vol. 66, no. 2, Fev 1978.

50 - A.K.Jones, R.J.Chansler, I.Durham, P.H.Feiler, D.A.Scelza, K.Schwans e S.R.Vegdahl, "Programming Issues Raised by a Multiprocessor", Proc. IEEE, vol. 66, no.2, Fev. 1978.

51 - P.H. Enslow Jr. (ed.), "Multiprocessors and Parallel Processing", Comtre Corporation, John Wiley Sons, 10a. edição, 1974.

52 - P.Y. Chen, D.H.Lawric, P.C.Yew, D.A.Padua, "Interconnection Networks Using Shuffles", Computer, vol. 14, no. 12, pp.55-64, Dez 1981.

53 - D.D.Clark, K.T.Pogrdan, D.P.Reed, "An Introduction to Local Area Networks", Proc. of IEEE, vol. 66, no. 11, pp.1497-1517, Nov. 1978.

54 - J.Gracia, L.C.Souza, "Rede Local Aplicada à Supervisão e Controle de Sistemas Elétricos", IX Congresso Nacional de Informática, RJ - Brasil, Out. 1982.

55 - R.M.Metcalfe, D.R. Boggs; "Ethernet: Distributed Packet Switching for Local Computer Networks", Comm. of ACM, vol. 19, no. 7, pp.395-404, Jul 1976.

56 - J.Gracia; "Desenvolvimento de uma Rede de Microprocessadores Aplicada a Supervisão de Sistemas Elétricos", Tese M.Sc. COPPE-UFRJ, 1980.

57 - D.A. Menascé, D. Schwkabe; "Redes de Computadores - Aspectos Técnicos e Operacionais", Terceira Escola de Computação, Dept. de Informática PUC/RJ, 1982.

58 - Softech Microsystems; "UCSD p-System and UCSD Pascal. Users Manual. Version IV.0",Second Edition, Jan 1981.

59 - Softech Microsystems; "UCSD p-System and UCSD Pascal. Installation Guide. Version IV.0", First Edition, Fev 1981.

60 - Softech Microsystems; "UCSD p-System and UCSD Pascal. Internal Architecture Guide. Version IV.0", First Edition, Mar 1981.

61 - UCSD - Institute for Information Systems; "UCSD (Mini-Micro Computer) Pascal. Version I.5", Second Printing, Dez 1978.

62 - Intel Co., "Intel Multibus Specification", Manual order no. 9800683, 1978.

63 - J. Barthmaier, "Intel Multibus Interfacing", Application note AP-28A, OEM Microcomputer Systems Applications, Jan 1979.

64 - Intel Co., "Introduction to the APX 286, Manual order no. 210308-001, Fev 1982.

65 - Intel Co., "SBC 86/12A Single board Computer Hardware Reference Manual", Manual order no. 9803074-2, Mar 1981.

66 - D.L.Collins, C.M.Collins, "Memory Management Chip Masters Large Data Bases", Electronic Design, pp.115-121, vol.29, no. 17, Ago 1981.

67 - R. Mateoslan, "Segmentation Advances AC Memory Addressing", Electronic Design, vol. 29, no. 4, pp. 155-162, Fev 1981

68 - D. Bursky, "Microprocessors - 4 to 32 bit - Push Back Performance Limits", Electronic Design, vol. 28, no. 24, pp.109-115, Nov. 1980.

69 - V.C. Barbosa, L.A. Terry, J. Motta; "Uma Proposta para Estender o Sistema Pascal UCSD a Processamento Concorrente", Relatório Técnico do CEPEL, no. 515/82, Ago 1982.

70 - A.L. Bogado; "Multi-Microprocessor Terminal for Electric Energy Systems Monitoring", IEEE Industrial Electronic and Control Instrumentation Group, Philadelphia, USA, Mar 1978.

71 - J. Motta, "Projeto e Construção de um Operador de Dados Digitais" Projeto Final de Curso, Depto. Engenharia Eletrônica UFRJ, Dez 1977.

72 - A.L.Bogado, L.A.Terry, O.Appel, M.Moszkowicz, R.S.Costa, J.Gracia, J.Motta, H.G.Andrade; "Centros de Supervisão para Sistemas Elétricos", V SNPTEE, Recife, Brasil, 1979.