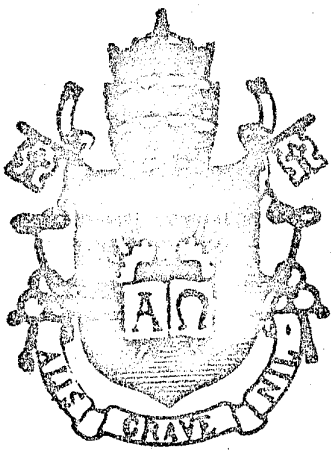


PUC

Série: Monografias em Ciência da Computação,
No. 6/90

HIERARQUIAS DE MEMORIA

Solon B. Silva

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP-22453
RIO DE JANEIRO - BRASIL

PUC Rio - Departamento de Informática

Série: Monografias em Ciência da Computação, Nº 6/90
Editor: Paulo A. S. Veloso Julho, 1990

HIERARQUIAS DE MEMÓRIA

Solon B. Silva

Exame de Qualificação (Doutorado) apresentado ao Prof. D. Menascé

Responsável por publicações:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC RIO, Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453 - Rio de Janeiro, RJ
BRASIL

Tel.: (021) 529-9386 TELEX: 31078 FAX: (021) 274-4546
BITNET: userrtlc@lncc.bitnet

Programa de Doutorado - PUC/RJ

HIERARQUIAS DE MEMÓRIA

Aluno: Solon Benayon da Silva

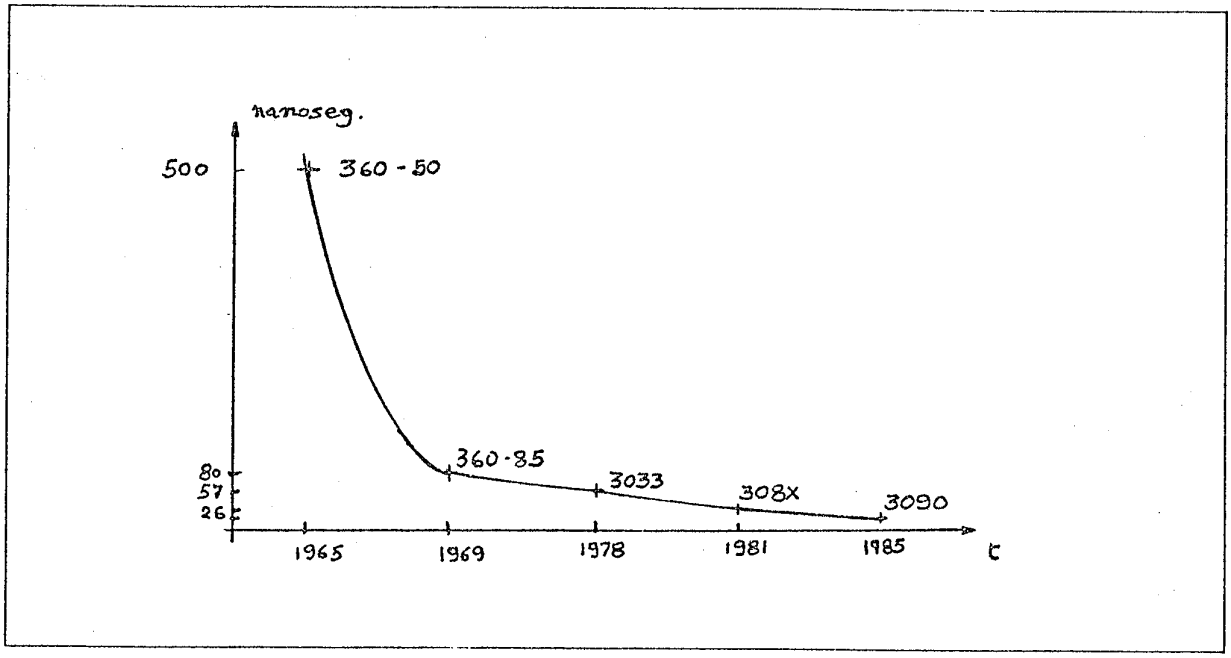
Orientador: Prof. Dr. Daniel A. Menascé

1ª Qualificação

INTRODUÇÃO

O desempenho global de um Sistema de Computação tem uma parcela significativa devida ao desempenho dos processadores nele utilizados. Os avanços da tecnologia microeletrônica vem permitindo a construção de processadores com desempenho cada vez maiores o que, por sua vez, acarreta o aparecimento de Sistemas de Computação cada vez mais eficientes e com maior desempenho.

Uma das formas de caracterizar o desempenho de um processador é através do tempo gasto por ele no processamento de uma operação unitária, tempo esse que se denomina ***ciclo do processador***. A Figura 1 mostra a evolução dos Sistemas de Computação através de seus tempos de ciclo de processamento (CONT68, TUCK86).



Evolução do tempo de ciclo de processamento

Para que um processador seja utilizado em seu nível máximo de desempenho, é necessário que os dados estejam a sua disposição, tão logo solicitados. Se isto for possível é evitada a ociosidade do processador que ocorreria caso este ficasse aguardando a chegada de dados necessários a continuação do processamento. Como os dados estão armazenados na memória, é sempre necessário dispender algum tempo para colocá-los a disposição do processador, tempo este correspondente ao tempo gasto na localização dos dados na

memória, somado ao tempo gasto no transporte desses dados entre a memória e o processador. Este tempo é denominado *ciclo de memória*. Também aqui a tecnologia vem permitindo alcançar ciclos de memória cada vez menores, conforme comprova a Figura 2 (CONT68, TUCK86).

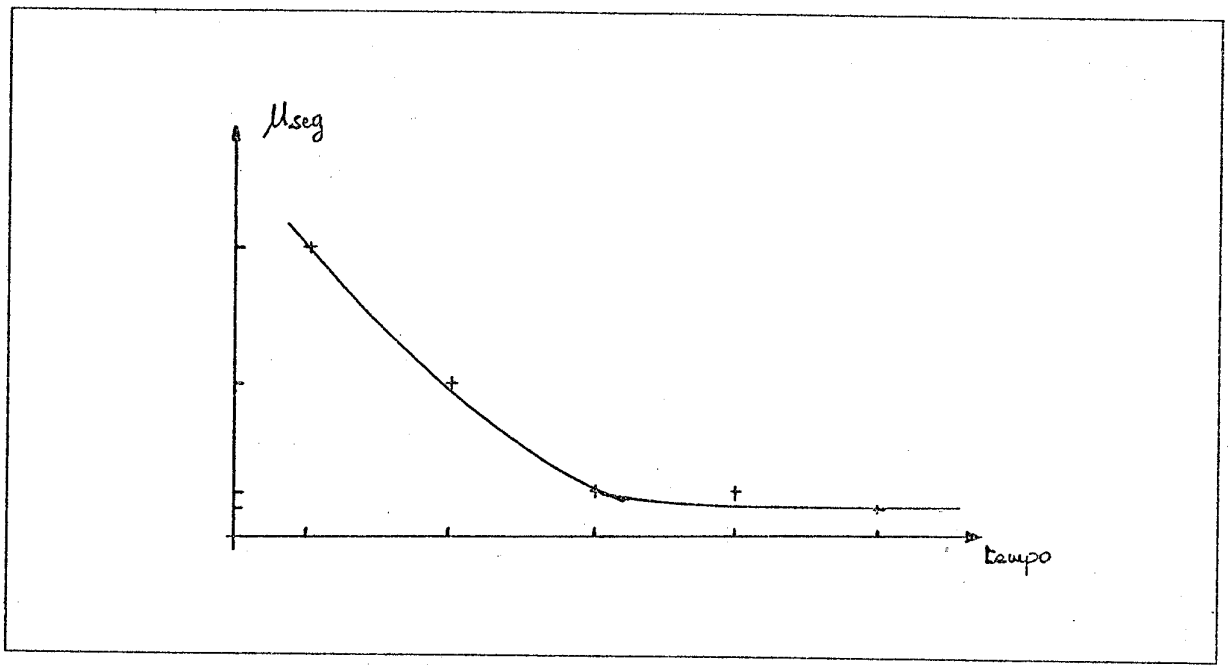


Figura 2 - Evolução dos Sistemas de Computação atraves de seus ciclos de memória

Uma comparação entre o tempo de ciclo apresentado pelo processador e o tempo de ciclo apresentado pela memória, para um mesmo Sistema de Computação, mostra uma grande discrepância entre os seus valores. Enquanto o tempo de ciclo do processador está na ordem de nanosegundos, o tempo de ciclo da memória é da ordem de microsegundos, diferença esta que se deve a utilização de tecnologias diferentes para a construção de memórias e para a construção de processadores.

HIERARQUIA DE ACESSO AOS DADOS

Uma situação ideal seria aquela em que a memória e o processador pudessem estar englobados em um mesmo substrato e, portanto, construidos com a mesma tecnologia. A utilização da mesma tecnologia garantiria a semelhança entre seus tempos de ciclo e, além do mais, a proximidade física minimizaria o tempo gasto no transporte dos dados. No entanto, tal situação ainda não é viável uma vez que os Sistemas de Computação atuais utilizam grandes capacidades de memória e a construção de memórias de grande capacidade, utilizando a mesma tecnologia aplicada na construção do processador, não é

viável porque leva a ocupação de grandes espaços físicos, elevados custos, além da necessidade de dissipação de grandes quantidades de calor.

Para contornar esse problema, surge a idéia de fazer uma composição entre as diversas tecnologias disponíveis aproveitando as vantagens apresentadas por cada uma delas e contornando suas limitações. Para tal, a memória é dividida em partes ou camadas, tendo cada camada como base a relação entre o seu custo de construção e o tempo de acesso aos dados ali contidos. Essas camadas assim construidas definem uma ***hierarquia*** que determina a ordem de acesso e movimentação dos dados. A Figura 3 mostra as camadas e a hierarquia de acesso aos dados a partir do processador.

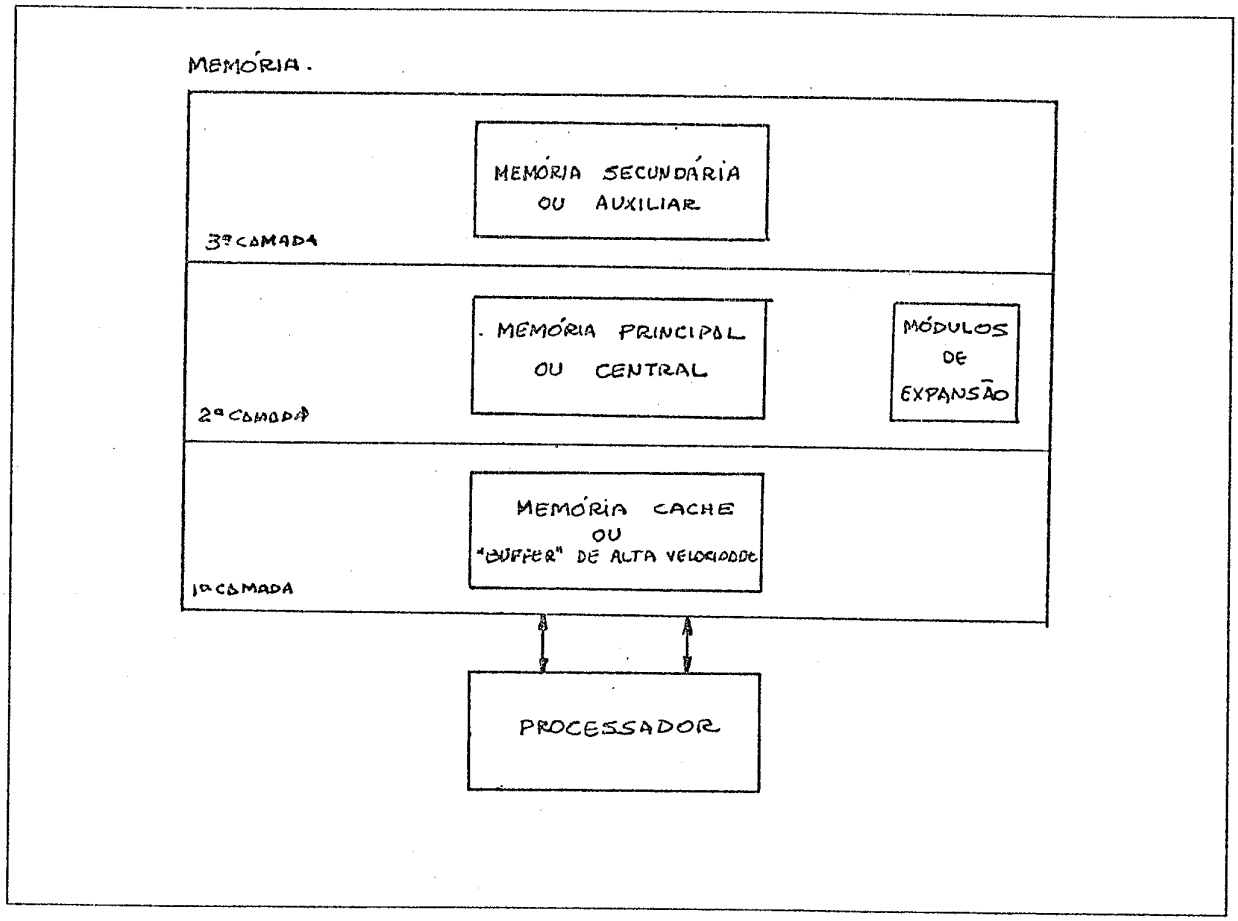


Figura 3 - Divisão da memória em camadas segundo uma hierarquia

A primeira camada, a mais próxima do processador, é construida com a mesma tecnologia utilizada na construção do processador. Devido a limitações de espaço e custo essa camada apresenta uma pequena capacidade de armazenamento se comparada com a capacidade das outras. No entanto, por ser construida com a mesma tecnologia utilizada pelo processador, apresenta um tempo de ciclo equivalente ao dele. por ele. Essa camada

é denominada *Memória Cache* ou "buffer" de alta velocidade. Os sistemas com um único processador podem ter uma ou várias Memórias Cache. Nos sistemas com vários processadores, cada um deles pode possuir a sua própria Memória Cache ou então, eles podem compartilhar uma única Memória Cache.

A segunda camada, denominada *Memória Principal* ou *Memória Central* vem em seguida a primeira camada e é construida com uma tecnologia desenvolvida especificamente para a construção de memórias. Esta tecnologia apresenta um tempo de ciclo elevado se comparado com o tempo de ciclo da Memória Cache. No entanto, ela permite que se construam memórias com grande capacidade e a custos reduzidos. Além disso, técnicas de acesso e endereçamento de dados permitem que a Memória Principal possa se expandir em módulos adicionais o que oferece um significativo potencial de crescimento.

Finalmente, a terceira camada, denominada *Memória Secundária* ou *Memória Auxiliar*, corresponde a memórias constituidas por dispositivos periféricos à Unidade Central de Processamento, como por exemplo as Unidades de Disco Magnético. A tecnologia utilizada na construção dessa camada permite a obtenção de enormes capacidades a custos razoáveis, mas com tempos de ciclos maiores do que os apresentados nas camadas inferiores da hierarquia.

Em termos de funcionamento do Sistema de Computação, um dado pode ser encontrado em qualquer uma das camadas. Quando o processador necessita de um dado, ele vai inicialmente procurá-lo na Memória Cache. Se o dado é ali encontrado, então uma cópia é transferida ao processador em um tempo compatível com o ciclo da Memória Cache que, como já vimos, é equivalente ao ciclo do processador. Se o dado lá não é encontrado, o sistema de controle vai então procurá-lo na Memória Principal. Caso ele seja alí encontrado, então uma cópia é transferida para o processador e também para a Memória Cache. Finalmente, se o dado somente estiver na Memória Auxiliar, ele é copiado para a Memória Principal e daí então passado ao processador e copiado também na Memória Cache.

Com relação ao desempenho do sistema, deve ser levado em consideração que todos esses movimentos se dão nos respectivos tempos de ciclo associados a cada uma das camadas. Tomando como exemplo o Sistema IBM 3090, o ciclo de sua Memória Cache está na ordem de nanosegundos, o ciclo de sua Memória Principal é da ordem de microsegundos e o ciclo da Memória Auxiliar é da ordem de milisegundos (COHE89). O desempenho do sistema é determinado pela média dos tempos de ciclo gastos na busca e no armazenamento dos dados localizados em cada uma das camadas. Quanto maior for a

concentração de dados nas camadas mais baixas da hierarquia, menor sera o tempo médio de ciclo da memória.

Como as camadas mais baixas da hierarquia e, principalmente a Memória Cache, tem capacidades menores de armazenamento, a maior concentração de dados é obtida através de sua movimentação para aquelas camadas, movimentação esta baseada em uma previsão de sua utilização dos dados pelo processador.

Essa previsão é possível porque a execução de programas acarreta referências a endereços da memória seguindo um determinado perfil, próprio de cada programa e caracterizado por uma propriedade denominada ***localidade.***

LOCALIDADE

A propriedade denominada localidade, apresentada pelos programas, determina que os dados por eles referenciados estão agrupados em determinadas regiões da Memória Principal. A propriedade da localidade apresenta dois aspectos: um temporal e um espacial.

O aspecto temporal de localidade, ou seja, a localidade no tempo, significa que o dado que está sendo referenciado naquele momento tem grandes possibilidades de ser novamente referenciado em um futuro próximo. Este tipo de comportamento pode ser observado nos "laços" de programa onde cada conjunto de dados é instruções são reutilizados durante um determinado intervalo de tempo.

O aspecto espacial, ou seja, a localidade no espaço determina que, em um futuro próximo, as referências feitas por um programa estão localizadas em endereços próximos ao endereço que está sendo referenciado naquele momento. Este tipo de comportamento pode ser observado na referência a dados relacionados entre si tais como variáveis, arranjos etc. e também na execução de instruções seqüenciadas.

O conhecimento obtido através da propriedade de localidade permite que uma grande parte das futuras referências possam ser previamente colocadas na Memória Cache de tal forma a que, quando forem necessárias, possam ali serem encontradas. Com isso, o tempo médio de ciclo de memória pode ser trazido a um valor próximo do valor do tempo de ciclo apresentado pela Memória Cache.

"HITS" E "MISSES" DA MEMÓRIA

Obviamente é impossível prever todas as referências futuras a memória que serão feitas pelo processador. O que se dispõe hoje são de algoritmos, mecanismos e técnicas que procuram não somente posicionar os possíveis dados a serem refenciados, nas camadas inferiores da memória, mas também otimizar os movimentos de dados entre as camadas. Cabe ao Sistema de Controle implementar essas ferramentas, sempre de uma forma transparente ao usuário.

Quando o processador necessita um dado, ele vai buscá-lo inicialmente na Memória Cache. Se o dado lá é encontrado diz-se ter ocorrido um ***Cache Hit***. Se o dado não é ali encontrado, diz-se ter ocorrido um ***Cache Miss***. Igualmente, se um dado for procurado na Memória Principal e lá não for encontrado, dizemos ter ocorrido um ***Page Faut*** ou um ***Page Hit*** no caso inverso.

O desempenho do sistema pode ser avaliado através do número de hits e misses ocorridos na Memória Cache. A efetividade da Memória Cache é usualmente descrita através da relação entre o número de "Cache Hits" e o total de referências feitas a memória. Tipicamente, esse valor está acima de 98% e, embora melhoras nesse valor possam parecer insignificantes, o impacto que elas causam no desempenho do sistema é significativo. Uma variação de 98% para 99% significa uma queda na taxa de "misses" de 2% para 1%. Essa pequena variação de 1%, no entanto, significa que o número de acessos a Memória Principal é reduzido a metade e que o atrazo total devido a memória será dividido por dois. Essa variação pode ser melhor avaliada através do ***tempo médio de ciclo de memória***, dado pela fórmula:

$$t_{ef} = h\ t_{cache} + (1-h)\ t_{princ}$$

onde:

t_{ef}	tempo efetivo médio de acesso à memória
h	relação de "Cache Hits"
T_{cache}	tempo de acesso à Memória Cache
T_{princ}	tempo de acesso à Memória Principal

Cabe também ao Sistema de Controle a decisão entre a continuação de uma tarefa ou a carga de uma nova tarefa quando da ocorrência de um "miss" que causa uma ociosidade no processador até que os dados lhe sejam entregues.

ORGANIZAÇÃO DOS DADOS NA MEMÓRIA

Os dados estão armazenados na memória segundo uma determinada organização e seqüência que visa a facilitar e otimizar a sua localização. Embora os dados estejam caracterizados individualmente pelo endereço em que estão armazenados na Memória Principal, eles são movimentados em grupos entre as diversas camadas da memória.

Os dados armazenados na Memória Cache estão organizados em linhas. Uma linha da Memória Cache é formada por vários bytes que correspondem a endereços seqüenciais. O tamanho dessas linhas é uma decisão de projeto da Memória Cache e leva em consideração a eficiência com que os dados possam alí ser encontrados. O movimento de dados entre a Memória Principal e a Memória Cache ocorre sempre em grupos de bytes seqüenciados, correspondentes a linha da Memória Cache.

Já a Memória Auxiliar está organizada em termos de blocos ou páginas, também correspondentes a um conjunto de endereços seqüenciais. O movimento de dados entre a Memória Auxiliar e a Memória Principal é feito em grupos, que correspondem aos blocos ou páginas. O tamanho desses blocos é definido de acordo com as necessidades do Sistema de Controle. Tipicamente, cada bloco tem um tamanho de 4 K Bytes.

LOCALIZAÇÃO DOS DADOS NA MEMÓRIA

A divisão da memória em camadas é uma divisão conceitual, não implicando portanto em nenhuma limitação quanto ao posicionamento da Memória Cache seja junto a Memória Principal, seja junto ao processador. Avanços tecnológicos já permitem que a Memória Cahe seja construida no mesmo chip do processador (PERR89) com todas as vantagens inerentes a esse posicionamento.

Em termos operacionais, os dados são localizados através do endereço que ocupam na Memória Principal. O processador, ao necessitar um determinado dado o faz através de um endereço que é colocado em seu registrador de endereçamento. A extensão da faixa de endereçamento que pode ser utilizada vai depender do tamanho desse registrador. Os sistemas de computação atuais utilizam registradores de endereçamento de 32 bits (31 bits de endereço mais um bit de controle). Com 31 bits é possível endereçar cerca de 4,3 Gigabytes.

Pelo conceito de Memória Virtual, só uma parte dos dados possíveis de serem referenciados está na Memória Principal, estando a parte restante armazenada na Memória Secundaria ou Auxiliar. O endereço gerado pelo processador é um endereço

virtual que pode referenciar todos os dados armazenados no Sistema de Computação. O endereço virtual é dividido em campos que podem ser vistos na Figura 4.

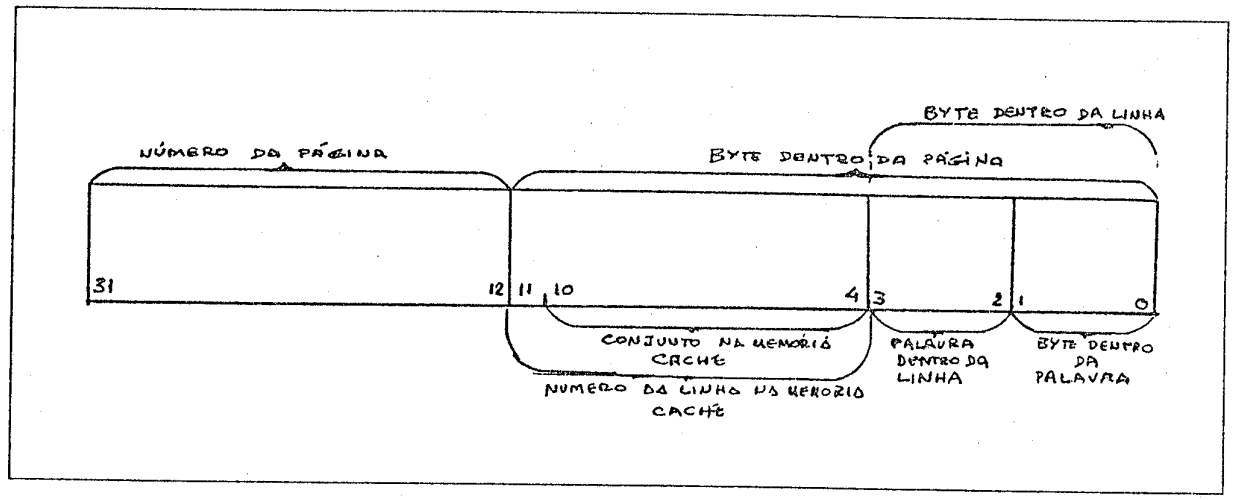


Figura 4 - Endereço Virtual e seus respectivos campos

O campo da esquerda, correspondente aos bits de mais alta ordem, define o número da página enquanto o campo da direita, correspondente aos bits de mais baixa ordem, define o endereço do dado dentro de cada bloco ou página. É possível subdividir o campo de endereço do dado de tal forma a que cada sub-campo passe a representar os endereços das linhas e os endereços de cada byte ou conjunto de bytes dentro de cada linha.

O *endereço real* é o endereço onde o dado está armazenado na Memória Principal. Como os dados são trazidos da Memória Auxiliar para a Memória Principal em blocos e ali posicionados segundo critérios próprios do Sistema de Controle, os endereços reais tem que ser calculados a partir do endereço inicial de carga do bloco na Memória Principal. Esse endereço inicial de carga do bloco na Memória Principal é chamado endereço base e o endereço do dado dentro de cada bloco é chamado deslocamento. Assim, o endereço real é calculado a partir do endereço virtual somando o endereço do byte dentro do bloco ao endereço inicial de carga do bloco na Memória Principal.

Um aspecto importante apresentado pelos endereços reais e virtuais diz respeito ao fato de que, enquanto o processador faz referência a um dado através de seu endereço virtual, a referência a Memória Cache é geralmente feita através de seu endereço real, o que obriga a uma tradução entre um e outro tipo de endereço. Essa e outras operações auxiliares podem ser melhor visualizadas através da Figura 5 onde está representado o processador, a Memória Cahe e alguns circuitos adicionais que executam aquelas operações.

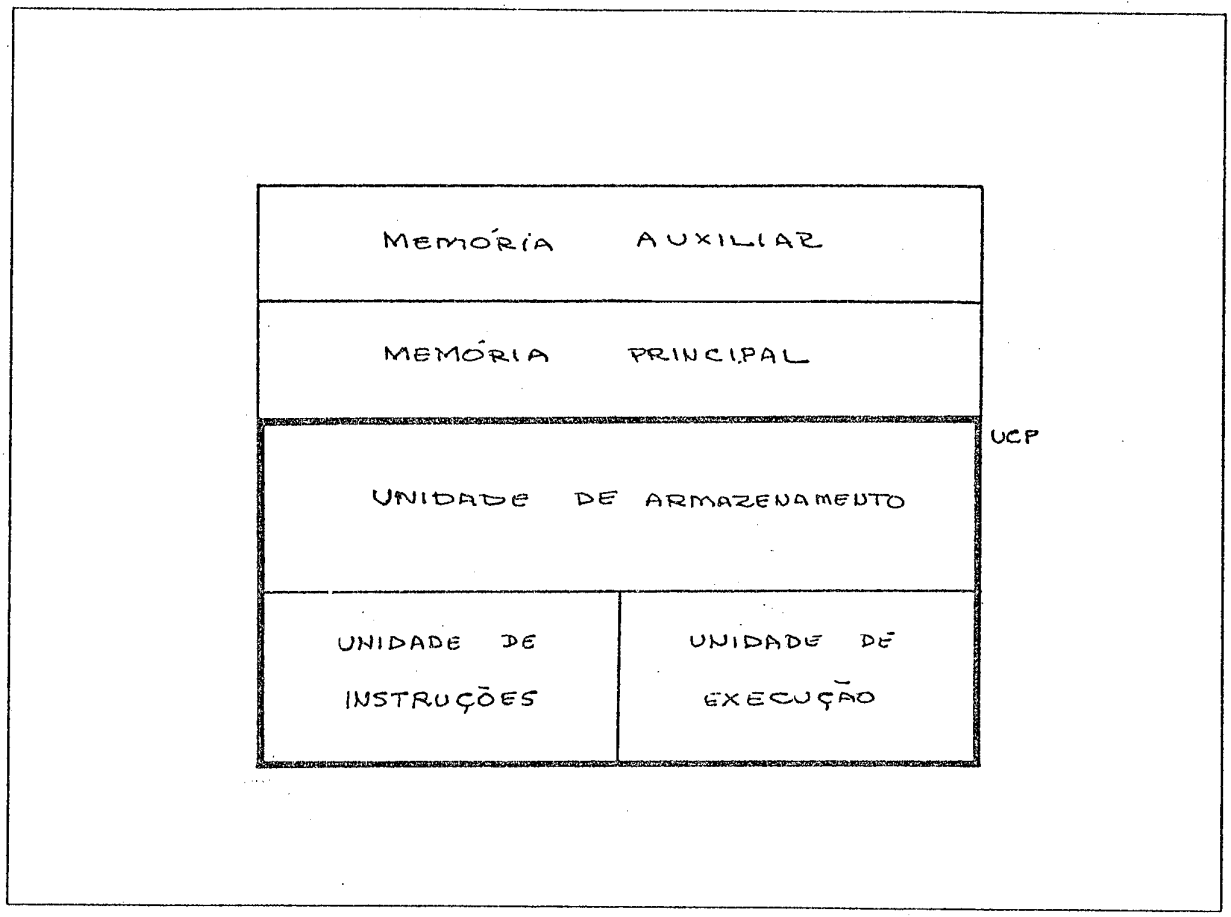


Figura 5 - Esquema típico de uma Unidade Central de Processamento

Também conceitualmente, uma Unidade Central de Processamento pode ser dividida em três partes: unidade de instruções, unidade de execução e unidade de armazenamento.

A unidade de instruções é responsável pela carga e decodificação das instruções, a unidade de execução contém os circuitos lógicos e aritméticos para executar as instruções e a unidade de armazenamento provê o interface entre a unidade de instruções e a unidade de execução.

A unidade de armazenamento por sua vez contem várias partes ou funções sendo a mais importante, a Memória Cache (1ª camada da hierarquia vista anteriormente). Existe também um circuito ***tradutor*** que traduz os endereços virtuais produzidos pelo processador em endereços reais da Memória Principal e um circuito denominado "***Translation Lookaside Buffer - TLB***" cuja função é armazenar os pares <endereço virtual-endereço real> mais recentemente utilizados. A Figura 6 mostra com mais detalhes, a unidade de armazenamento.

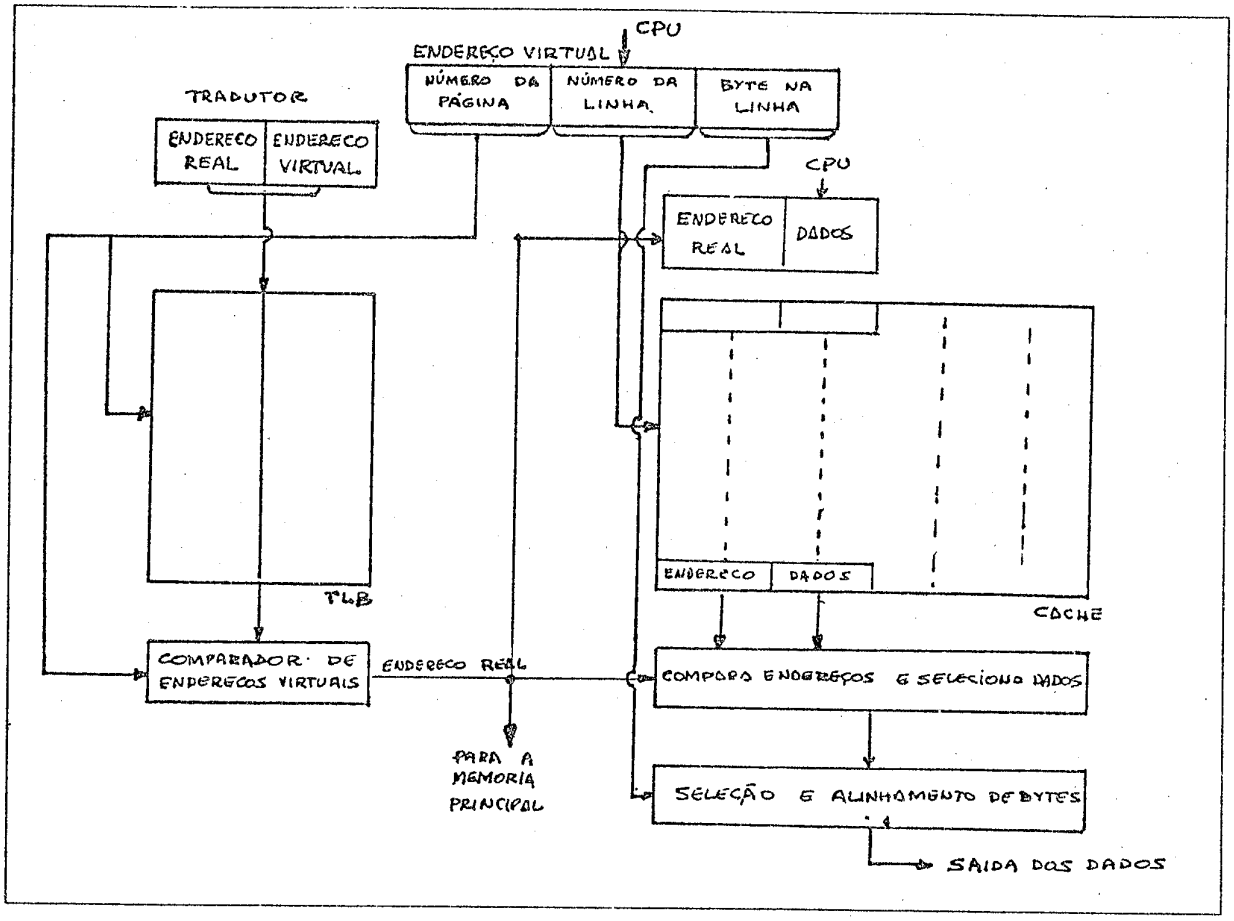


Figura 6 - Relacionamento típico entre a Memória Cache e o "TLB"

A operação se inicia com a chegada de um endereço virtual vindo do processador. Esse endereço virtual é passado diretamente ao TLB onde é feita a procura do par <endereço virtual-endereço real> que a ele corresponda. Ao mesmo tempo, o campo do endereço virtual correspondente a linha e passado à Memória Cache e a linha é colocada em um circuito comparador. Se o endereço virtual é encontrado no TLB, o endereço real resultante é então comparado com o valor encontrado no circuito comparador para verificar se a linha se encontra na Memória Cache. Caso a linha esteja na Memória Cache, os dados requeridos são selecionados através das informações contidas no campo de bytes do endereço virtual e então enviados ao processador.

Se o endereço virtual não é encontrado no TLB, o circuito tradutor é acionado e, através da consulta a tabelas internas ao sistema, o endereço real é calculado. O par assim formado vai para o TLB, continuando o processo da forma descrita anteriormente.

Caso ocorra um "miss", ou seja, a linha correspondente não seja encontrada na Memória Cache, então o endereço real da linha procurada é passado à Memória Principal. Em

resposta, a linha requerida é copiada na Memória Cache que, a esta altura já terá liberado uma outra linha para dar lugar a linha nova. Essa linha recebida é também colocada no circuito comparador de tal forma a enviar mais rapidamente os dados solicitados pelo processador.

A referência a Memória Cache feita diretamente através de endereços virtuais também é possível e apresenta como vantagem a economia de tempo resultante da eliminação da tradução de endereços. Como desvantagem, apresenta a ocorrência de sinônimos que podem aparecer nas seguintes situações:

- quando dois programas diferentes, utilizando um mesmo processador, compartilham blocos ou páginas de dados
- quando um programa solicita ao Sistema de Controle a atribuição de diferentes endereços virtuais para um mesmo endereço real
- quando um dispositivo de entrada/saida, usando endereços reais, faz referência a uma mesma região da Memória Principal que esta sendo acessada por um programa
- quando um programa, carregado em um processador, compartilha a mesma área da Memória Principal com outro programa, carregado em outro processador.

O problema dos sinônimos pode ser solucionado através da introdução de um circuito lógico com a função de mapear os endereços reais de volta para os endereços virtuais que o geraram. Este circuito se denomina *"Reverse Translation Buffer - RTB"*.

MOVIMENTAÇÃO DOS DADOS

Para serem processados, os dados e as instruções devem estar na Memória Principal ou na Memória Cache onde serão localizados através de seu endereço real. No início de funcionamento do Sistema de Computação, as instruções e os dados estão armazenados na Memória Auxiliar. Cabe ao Sistema de Controle colocá-los na Memória Principal para início do processamento. Nessa fase, a Memória Cache ainda está vazia e portanto, todas as referências iniciais para a memória, feitas pelo processador, vão resultar em "Cache Misses". Essas referências serão então satisfeitas através da Memória Principal e a medida em que instruções e dados são referenciados e entregues ao processador, são tambem simultaneamente copiados na Memória Cache. Essa cópia simultanea para a Memória Cache se dá não somente com o endereço que está sendo referenciado naquele momento mas também com os endereços subsequentes correspondentes a uma linha inteira da Memória Cache. Com esse procedimento, a probabilidade de ocorrência de um "Cache Hit" na próxima referência vai crescendo uma vez que a probabilidade dela ser satisfeita pela Memória Cache é aumentada.

Este problema verificado quando da entrada de uma tarefa em processamento tem um impacto significativo no desempenho do sistema pois até que o sistema atinja o seu estado estacionário, a taxa de "Cache Misses" é alta. Esta situação se agrava ainda mais quando se considera um funcionamento dinâmico, com várias tarefas sendo processadas em seqüência e cada uma com sua fatia de tempo de processamento previamente determinadas pelo Sistema de Controle. Neste caso, quando uma tarefa entra em processamento, a sua taxa de "cache Misses" é alta e, em conseqüência, o desempenho do sistema é baixo. Quando ela chega a alcançar o seu estado estacionário e o sistema vai melhorando o seu desempenho, o tempo de processamento da tarefa termina e uma nova tarefa entra em processamento, reiniciando-se o processo anteriormente descrito e voltando o Sistema a apresentar um baixo desempenho. A taxa de "Cache Misses" apresentada por um Sistema quando sua Memória Cache está vazia é conhecida como em situação de *"cold start"* e a tada de "Cache Misses" quando a Memória Cache já se encontra preenchida com dados relativos a tarefa em processamento é conhecida como taxa em *"warm start"*.

Um outro aspecto importante diz respeito a atualização dos dados na Memória Principal. Quando um dado é processado, é possível que, em decorrência desse processamento, o dado assuma um novo valor. Essa atualização deve então ser devolvida à Memória Principal de tal forma a que futuros acessos a este dado já assumam o seu novo valor. O problema que surge diz respeito a como atualizar o dado que, após ser referenciado pelo processador, vai estar simultaneamente na Memória Principal e na Memória Cache.

Duas filosofias (COHE89) são utilizadas na atualização de dados em sistemas com Memória Cache: Na primeira, denominada *"store-through"* ou tambem *"copy-through"*, a atualização dos dados é feita simultaneamente na Memória Cache e na Memória Principal.

Na segunda filosofia de atualização, denominada "**store-in**" ou tambem *"copy-back"*, as atualizações são feitas somente na Memória Cache. Essas atualizações são refletidas na Memória Principal somente quando o dado é removido da Memória Cache através de um mecanismo de realocação.

A vantagem da filosofia "store through" é a simplicidade. Uma vez que todas as atualizações na Memória Principal são feitas imediatamente o que faz com que os dados já atualizados estejam imediatamente disponíveis a outros processadores que acessam também a Memória Principal.

A vantagem da filosofia "store-in" é a redução no movimento de dados entre a Memória Cache e a Memória Principal, isto porque as mudanças somente são escritas na Memória Principal, quando requeridas. No entanto, o Sistema de Controle necessita implementar mais funções de controle uma vez que, caso outro processador necessite referenciar o mesmo dado, é necessário fazer uma verificação prévia no conteúdo das Memórias Cache dos outros processadores para determinar se elas contém uma versão mais atualizada para aquele dado.

A Memória Cache se comunica com o processador através de canais de comunicação. Esses canais são definidos pela sua ***largura*** em bytes ou seja, através do número de bytes que podem ser transportados de uma vez em uma movimentação de dados.

Essa largura do canal é tão importante para o desempenho da Memória Cache quanto o seu tempo de acesso. Para um desempenho adequado, a Memória Cache deve apresentar uma ***faixa de passsagem*** suficiente. A faixa de passagem de uma Memória Cache é definida como sendo a largura de faixa dividida pelo tempo de acesso.

MEMÓRIAS CACHE

As Memórias Cache são circuitos de memória que apresentam um tempo de acesso aos dados da ordem de grandeza dos tempos de ciclo dos processadores e, portanto, significativamente menores do que o tempo de acesso aos dados da Memória Principal. Pelo armazenamento prévio, na Memória Cache, de uma cópia dos dados que possivelmente serão referenciados em um futuro próximo, é possível diminuir o tempo médio de acesso a dados e instruções relativas a uma tarefa em processamento. Algumas limitações não permitem que se construam Memórias Cache com tamanho suficiente para armazenar todos os dados necessários ao processamento de uma tarefa, o que leva a criação de uma hierarquia de acesso aos dados nas memórias Cache, Principal e Auxiliar. Isto cria a necessidade de movimentação dos dados das camadas superiores para as camadas inferiores, movimentação esta que tem como objetivo entregar o dado, ao processador, no menor tempo possível.

Essas limitações fazem com que se procure otimizar a Memória Cache de forma tal a alcançar os seguintes objetivos:

- maximizar a probabilidade de atendimento a referências do processador pela Memória Cache (taxa de "Cache Hits")
- minimizar o tempo de acesso aos dados localizados na Memória Cache
- minimizar o atraso devido a ocorrência de "Cache Misses"
- minimizar o atraso devido a atualização da Memória Principal com a garantia de consistência entre as diversas memórias.

Tendo em vista esses objetivos, uma série de aspectos são levados em consideração quando do projeto da Memória Cache, aspectos esses que serão vistos a seguir:

ORGANIZAÇÃO DA MEMÓRIA CACHE

A Memória Cache está organizada em termos de ***linhas*** ou ***blocos*** que compreendem um ou vários endereços da Memória Principal. Os dados são copiados entre as memórias sempre em conjuntos correspondentes a uma linha ou bloco.

Cada linha da Memória Cache é identificada por uma chave que referencia todos os dados contidos naquela linha. Essa organização faz com que a Memória Cache se

comporte como uma ***memória associativa***. Isto significa que a localização de um dado é ser feita através da chave relativa a linha em que ele se encontra, pelo uso de uma função que permita o mapeamento entre o endereco do dado e aquela chave. Essa organização permite que a Memória Cache seja representada por um conjunto de pares <chave-linha> onde o conjunto de chaves forma o diretório que é pesquisado para a localização dos dados. Essa busca associativa leva um tempo relativamente grande devido ao tempo de propagação dos sinais elétricos através dos circuitos que implementam essa organização, se comparada com uma organização de memória onde o acesso aos dados seja feito em forma direta. No entanto, a sua facilidade de implementação justifica a limitação apresentada.

Para minimizar esse tempo de localização dos dados, costuma-se dividir as linhas em conjuntos que são então pesquisados independente e simultaneamente. Memórias com essa organização são denominadas ***memórias associativas em conjuntos*** e podem ser melhor visualizadas na Figura 8.

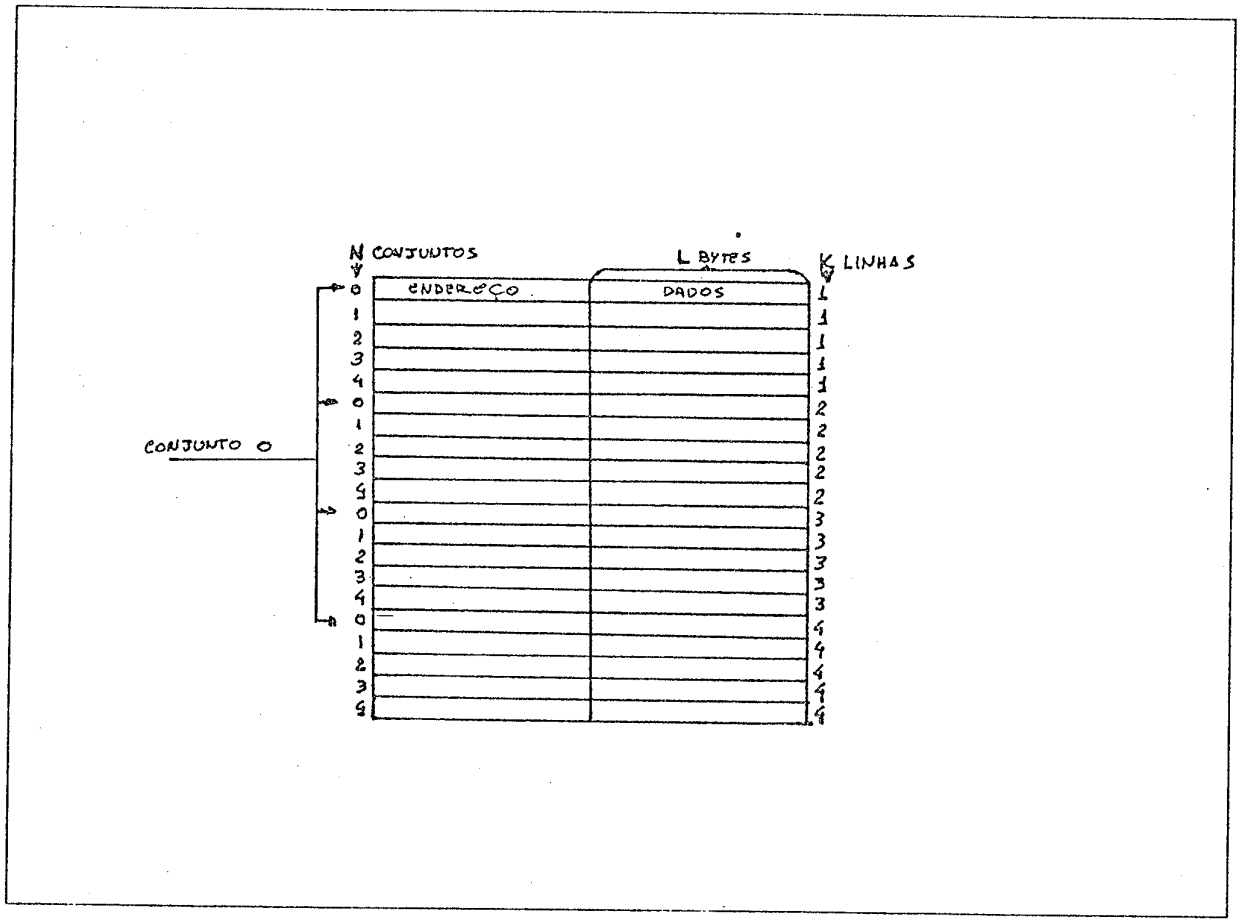


Figura 8 - Memória Cache Associativa em Conjuntos

Se considerarmos que cada linha tenha *L* bytes de comprimento, que as linhas estejam divididas em *N* conjuntos e que cada um desses conjuntos tenha *K* linhas, podemos dizer que a capacidade total da Memória Cache é igual ao produto *NKL*. Ao variar os valores de N, K ou L, vamos observar uma variação no desempenho da memória. Se L for igual a um byte, teremos uma chave para cada dado o que traz os benefícios do acesso direto mas acarreta a formação de um diretório muito grande, o que diminui a eficiência da Memória Cache. Se o valor de N for feito igual a um, o que representa a existência de um só conjunto, as linhas serão pesquisadas em seqüência, através de suas chaves. Memórias com essa organização são conhecidas como ***memórias totalmente associativas.*** Se K for feito igual a um, significa que cada conjunto será formado por apenas uma linha ou seja, a memória apresentará um acesso direto a cada linha, sendo então conhecida como ***memória de mapeamento direto.***

CARGA DA MEMÓRIA CACHE

O procedimento normal de preenchimento da Memória Cache é aquele em que uma linha é buscada na Memória Principal e copiada para a Memória Cache sempre que ocorre um "Cache Miss". A esse procedimento se dá o nome de ***busca por demanda.*** O tempo de acesso aos dados referentes a um "Cache Miss" pode ser diminuído aplicando-se uma técnica denominada ***"load-through"*** onde, em decorrência de um "Cache Miss", os dados localizados na Memória Principal, são entregues diretamente ao processador, podendo a Memória Cache ser carregada simultaneamente ou em seguida a esse procedimento.

Como se tem como objetivo a diminuição da taxa de "Cache Misses", um procedimento indicado seria a busca de linhas na Memória Principal antes que elas fossem referenciadas, sendo para tal sugeridos vários algoritmos. No entanto, a sua aplicação requer certos cuidados uma vez que a colocação de linhas novas na Memória Cache, com a justificativa de que elas poderão vir a ser utilizadas, causa a retirada de outras linhas que ainda poderiam ser utilizadas resultando em um fenômeno conhecido como ***poluição de memória.***

Um dos algoritmos de busca prévia que apresenta um bom desempenho é o algoritmo denominado ***"one block lookahead"*** onde uma linha *i* referenciada credencia a linha *i + 1* como candidata a carga prévia na Memória Cache. Nesse algoritmo três opções podem ocorrer: na primeira, denominada ***"always prefetch"*** (SMIT82), sempre que a linha *i* é referenciada, a linha *i + 1* é carregada na Memória Cache. Na segunda opção, denominada ***"prefetch on misses"***, a carga prévia somente é realizada se a referência a linha *i* for em decorrência de um "Cache Miss" e, na terceira, denominada ***"tagged pre-***

fetch", um bit extra é adicionado a cada linha e seu valor controla o movimento de carga da linha (GIND87).

MAPEAMENTO DOS DADOS NA MEMÓRIA CACHE

A Memória Cache, como já se viu, não pode ser diretamente endereçada pelos usuários, decorrendo daí a necessidade de uma função que permita um mapeamento entre o endereço do dado na Memória Principal e a sua localização na Memória Cache. A forma mais utilizada para esse mapeamento é a conhecida como ***"bit selection"***, que está baseada em uma estrutura de endereçamento mostrada na Figura 9.

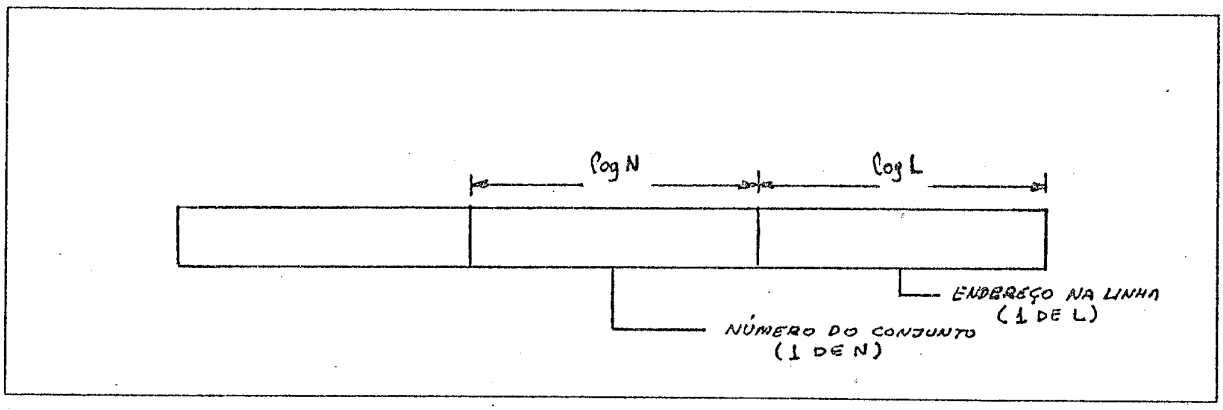


Figura 9 - Esquema de endereçamento para a função de mapeamento

Essa forma de mapeamento é obtida através de uma manipulação bastante simples no endereço real que é enviado à Memória Principal, quando da ocorrência de um "Cache Miss". Se considerarmos que o endereço tem um comprimento de "m" bits, os primeiros ($Log_2\ L$) bits menos significativos do endereço vão corresponder ao L bytes da linha. Os ($Log_2\ N$) bits seguintes do endereço vão corresponder aos N conjuntos pelos quais a Memória Cache é dividida. Essa forma de especificar os conjuntos vai permitir que haja um espalhamento das linhas adjacentes da Memória Principal ao longo de todos os conjuntos.

Finalmente, os (M - ($Log_2\ N$ + $Log_2 L$)) bits restantes vão corresponder então as chaves que caracterizam cada linha.

Com esse esquema de mapeamento, ao ocorrer um "Cache Miss", o endereço é enviado para a Memória Principal e é separado então um bloco de endereços que incluem aquele necessitado e mais alguns endereços que lhe sucedem, bloco esse correspondente a uma linha da Memória Cache. O campo correespondente ao conjunto vai então determinar em

qual conjunto o bloco será colocado e o campo restante correspondente a chave vai determinar em qual linha o bloco está armazenado.

LOCALIZAÇÃO DOS DADOS NA MEMÓRIA CACHE

A localização dos dados na Memória Cache se faz também com base no esquema de mapeamento indicado anteriormente. Essa localização pode ser melhor entendida com o auxílio da Figura 10.

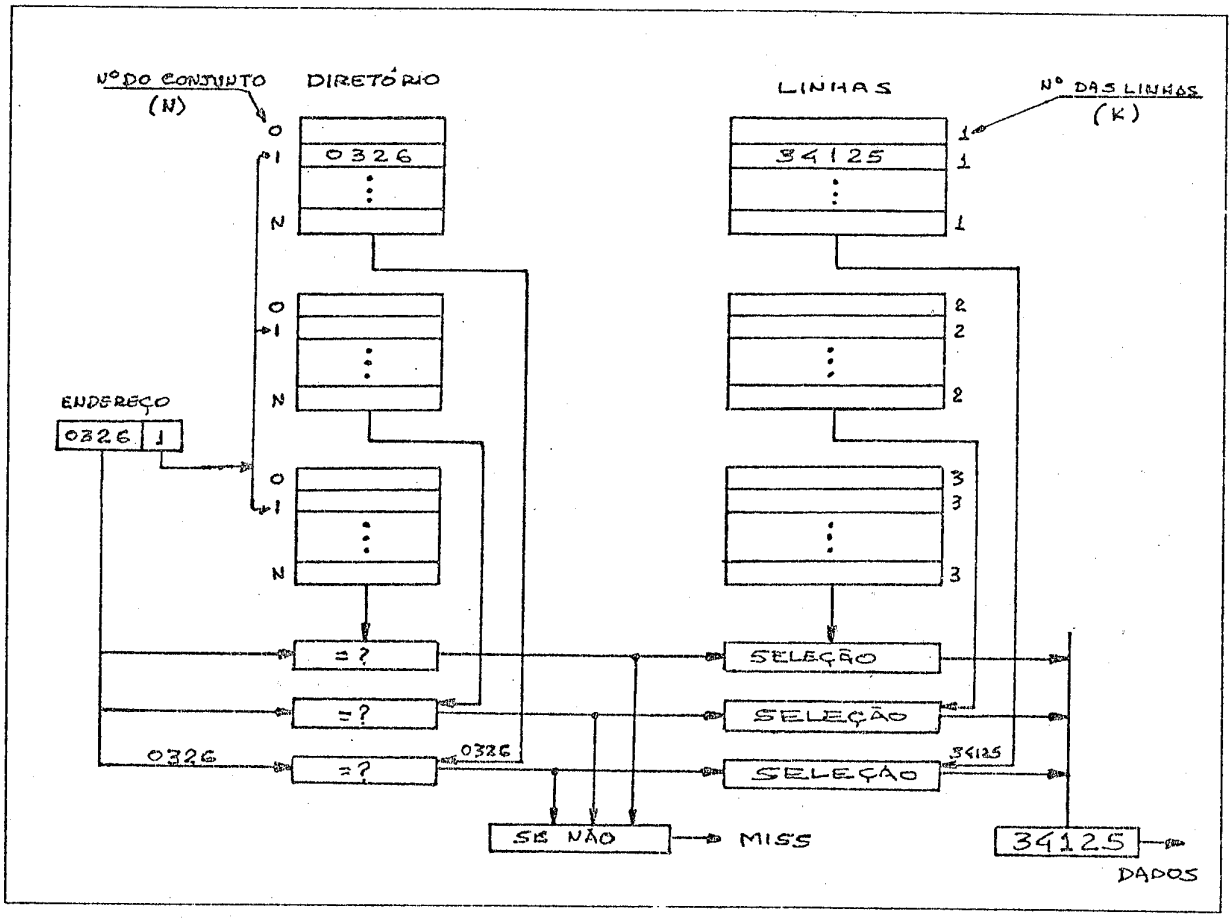


Figura 10 - Localização de dados na Memória Cache

Na Figura 10 é mostrado o exemplo de uma Memória Cache associativa por conjuntos, com um total de N conjuntos sendo cada conjunto formado por K linhas contendo cada uma L endereços (aqui considerados como um byte por endereço). Ao receber um endereço, a Memória Cache seleciona primeiramente o conjunto que corresponde àquele endereço. Ao ser feito isso, somente as K linhas correspondentes àquele conjunto serão simultaneamente lidas. No final dessa leitura, as K entradas do diretório assim disponível

são então comparadas com o campo do endereço correspondente a chave. Se dessa comparação, um casamento é encontrado, então a linha resultante é colocada no circuito de saida da Memória Cache e, uma operação simples de deslocamento com base no endereço do dado dentro da linha, vai permitir a obtenção do dado procurado.

SUBSTITUIÇÃO DE LINHAS

Quando o sistema de memória atinge o seu estado estável, a Memória Cache vai estar totalmente ocupada. Nessa condição, um "Cache Miss" obriga não somente a carga de uma linha da Memória Principal, mas também, a substituição de uma linha existente na Memória Cache, para a liberação do espaço necessário à carga da nova linha.

Vários algoritmos para a substituição de linhas foram propostos. Esses algoritmos propostos podem ser divididos em dois grupos de acordo com a sua referência ao uso ou não das linhas da Memória Cache.

Os algoritmos que levam em consideração o uso das linhas tomam como base a frequência de utilização das mesmas através do registro dessa utilização em algum campo específico para tal fim. Como exemplo desse grupo temos o algoritmo denominado ***LRU*** ("Least Recently Used") onde a linha utilizada menos recentemente é a candidata à substituição pela linha nova. No outro grupo temos os algoritmos que levam em conta outros fatores diferentes da utilização das linhas. Como exemplo deste grupo temos o algoritmo denominado ***FIFO*** ("First In First Out") onde a linha candidata à substituição segue a ordem de entrada na Memória Cache.

É importante notar que, em uma Memória Cache do tipo associativa por conjuntos, os algoritmos de substituição são implementados separadamente em cada conjunto, pois a carga de uma linha da Memória Principal obriga o seu posicionamento em um determinado conjunto.

Embora os vários algoritmos propostos apresentem vantagens, poucos são os facilmente implementáveis e, dentre estes últimos, os mais comumente encontrados são os algoritmos ***LRU***, ***FIFO*** e ***RAND*** (algoritmo aleatório ou pseudo-aleatório), com preferência para os dois primeiros.

Se compararmos o algoritmo ***LRU*** com o algoritmo ***FIFO***, o primeiro apresenta um desempenho melhor do que o segundo. Em termos de taxa de "misses" (SMIT87) o algoritmo ***LRU*** apresenta em média um desempenho superior em 12% se comparado com o algoritmo ***FIFO*** mas, no entanto, a sua implementação tem um maior custo.

MEMÓRIAS CACHE EM AMBIENTES DE MÚLTIPLOS PROCESSADORES

Até aqui foram abordados Sistemas de Computação compostos por um único processador com acesso a sua Memória Cache e a Memória Principal. Como esse acceso segue uma hierarquia previamente estabelecida, nenhuma precaução é necessária no que diz respeito a integridade dos dados.

Em Sistemas de Computação compostos por mais de um processador, problemas adicionais aparecem, de acordo com a configuração que possa ser montada. É possível ter-se uma única Memória Cache compartilhada pelos vários processadores ou então, uma Memória Cache para cada processador. Nesses ambientes, três problemas mais significativos aparecem: o primiero diz respeito ao tráfego de dados entre os processadores e as memórias. O segundo está relacionado com a *coerência* ou *consistência* entre as diversas Memórias Cache e o terceiro, está relacionado com o tempo de acesso aos dados.

Em um sistema de múltiplos processadores, a rede de comunicação que os interliga com a memória, tem no seu tráfego de dados um dos pontos de maior influência no desempenho do sistema. Em um sistema com uma única Memória Cache, cada acesso de dados por parte de um dos processadores, significa um incremento no tráfego de dados pela rede de comunicação. Já em um sistema onde cada processador tem a sua própria Memória Cache, somente os "Cache Misses" irão produzir incrementos no tráfego da rede de comunicação. Como a taxa de "Cache Miss" é um valor relativamente pequeno, é possível avaliar o impacto causado pela diminuição do tráfego de dados proporcionado por esta solução.

O segundo problema diz respeito a coerência entre os dados armazenados nas diversas memórias do sistema. Este problema aparece nos sistemas cuja configuração engloba vários processadores cada um com a sua própria Memória Cache e, principalmente, quando do uso de variáveis compartilhadas entre os diversos processos em processamento.

O terceiro problema diz respeito ao tempo de acesso aos dados da Memória Cache. Como vimos anteriormente, a minimização do tempo de ciclo da Memória Cache força o direcionamento do hardware para a construção de Memórias Cache o mais próximo possível do processador, com a tendência inclusive de posicioná-las no mesmo substrato do processador. Em sistemas que seguem essa tendência é natural a adoção de múltiplos processadores, cada um com a sua própria Memória Cache.

Dos três problemas apresentados, a coerência entre as diversas Memórias Cache é o que mais preocupa. Para se ter uma idéia mais clara deste problema, vamos imaginar o sistema da Figura 11, onde dois processadores, cada qual com a sua própria Memória Cache, compartilham uma mesma Memória Principal.

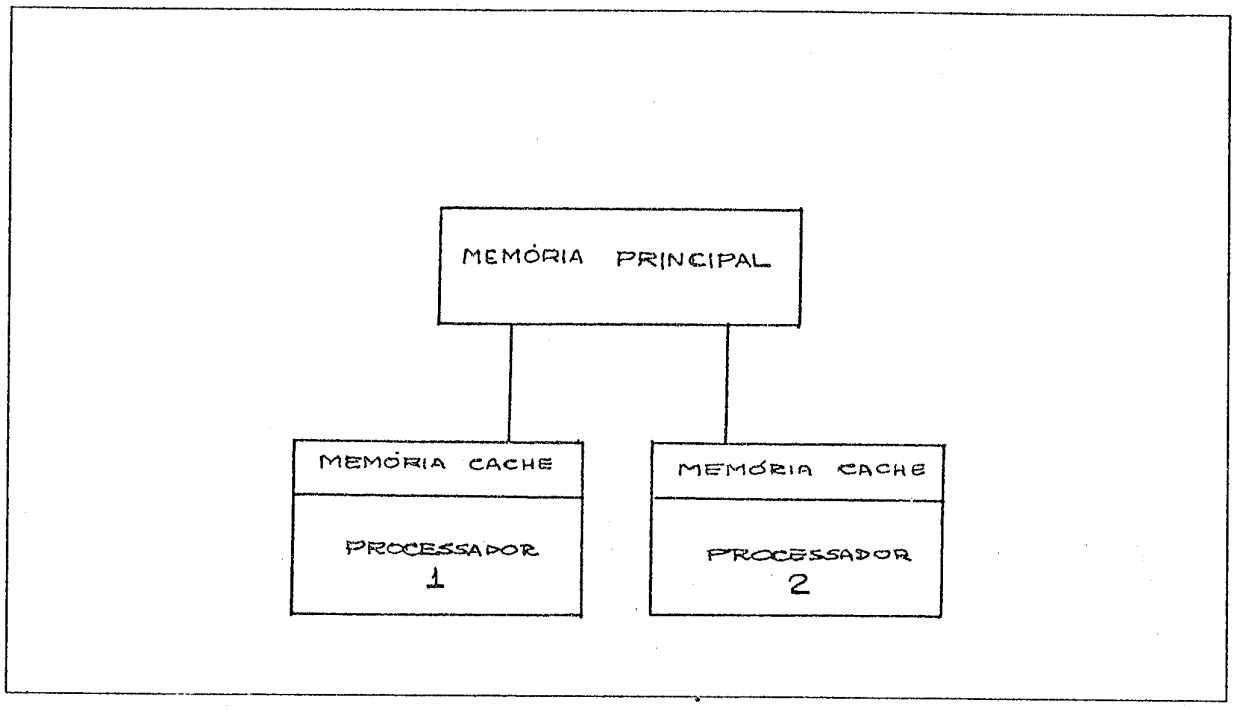


Figura 11 - Sistema com Múltiplos Processadores e Memórias Cache Próprias

Se o processador 1 faz referência a um dado que não está em sua Memória Cache, o dado vai ser apanhado na Memória Principal. Se este dado for uma variável compartilhada por outros processos, ao ser atualizado pelo processador que o necessitou, vai para a sua correspondente Memória Cache ou, para a Memória Principal em um tempo que depende da filosofia de atualização empregada seja ela "store-through" ou "copy-through". O problema de coerência aparece se, dentro deste intervalo de tempo, outro processador fizer referência ao mesmo dado na Memoria Principal, situação essa que fará com que ele referencie uma cópia desatualizada do dado.

Para assegurar a consistencia entre os dados não basta somente atualizar a Memória Principal. Vamos imaginar a situação em que o processador 1 atualiza um determinado dado e, imediata e simultaneamente, o coloca em sua Memória Cache e na Memória Principal. Se o processador 2 agora referencia o mesmo dado na Memória Principal, atualiza o seu valor e, igualmente ao processador 1, o devolve para a Memória Principal, não está assegurada a consistência dos dados porque, se o processador 1 referencia novamente o dado, ele agora se encontra em sua Memória Cache, mas o seu valor é

diferente do encontrado na Memória Principal, recentemente atualizado pelo processador 2.

Várias soluções para o problema de coerência entre Memórias Cache foram propostas todas, no entanto, limitadas a garantir a coerência para variáveis compartilhadas. Essas soluções variam no seu gráu de complexidade mas todas, sem dúvida, significam um fator preponderante de degradação do sistema.

Das soluções propostas, a mais simples é aquela em que as variáveis compartilhadas não são elegíveis a serem colocadas nas Memórias Cache. Nesse caso, qualquer referência a elas, sempre causa um "Cache Miss".

Uma solução relativamente simples, mas também impactante no desempenho do sistema corresponde a implementação de um protocolo onde todas as operações de leitura devidas a "Cache Misses" são feitas simultaneamente em todas as Memórias Cache e, todas as operações de escrita são também feitas em todas as Memórias Cache.

MEMÓRIAS CACHE EM AMBIENTES DE MULTIPROGRAMAÇÃO

Em ambientes de multiprogramação, duas ou mais tarefas competem pelo uso da mesma Memória Cache. Quando uma tarefa entra em processamento, as suas primeiras referências a memória ocasionam "Cache Misses". A medida em qua a Memória Cache vai sendo preenchida, a taxa de "Cache Misses" vai diminuindo até alcançar um valor estável. Quando o tempo de processamento alocado àquela tarefa termina, uma nova tarefa entra em processamento. Obviamente, esta tarefa não vai utilizar os dados da tarefa anterior e assim, os dados armazenados na Memória Cache são inúteis. Por isto, a taxa de "Cache Miss" sofre um abrupto aumento e, vai então diminuindo a medida em que novos dados vão sendo ali colocados, ate alcançar novamente o estado estacionário.

Este comportamento vai depender não só do tamanho da Memória Cache mas também da quantidade de dados referenciados por cada tarefa. Em um caso extremo é possível que uma Memória Cache de determinado tamanho consiga armazenar todos os dados necessários ao processamento de todas as tarefas que então serão processadas sem interferências entre elas.

Uma forma interessante de estudar esse problema é fazê-lo através do conceito de "footprint", ilustrado pela Figura 12.

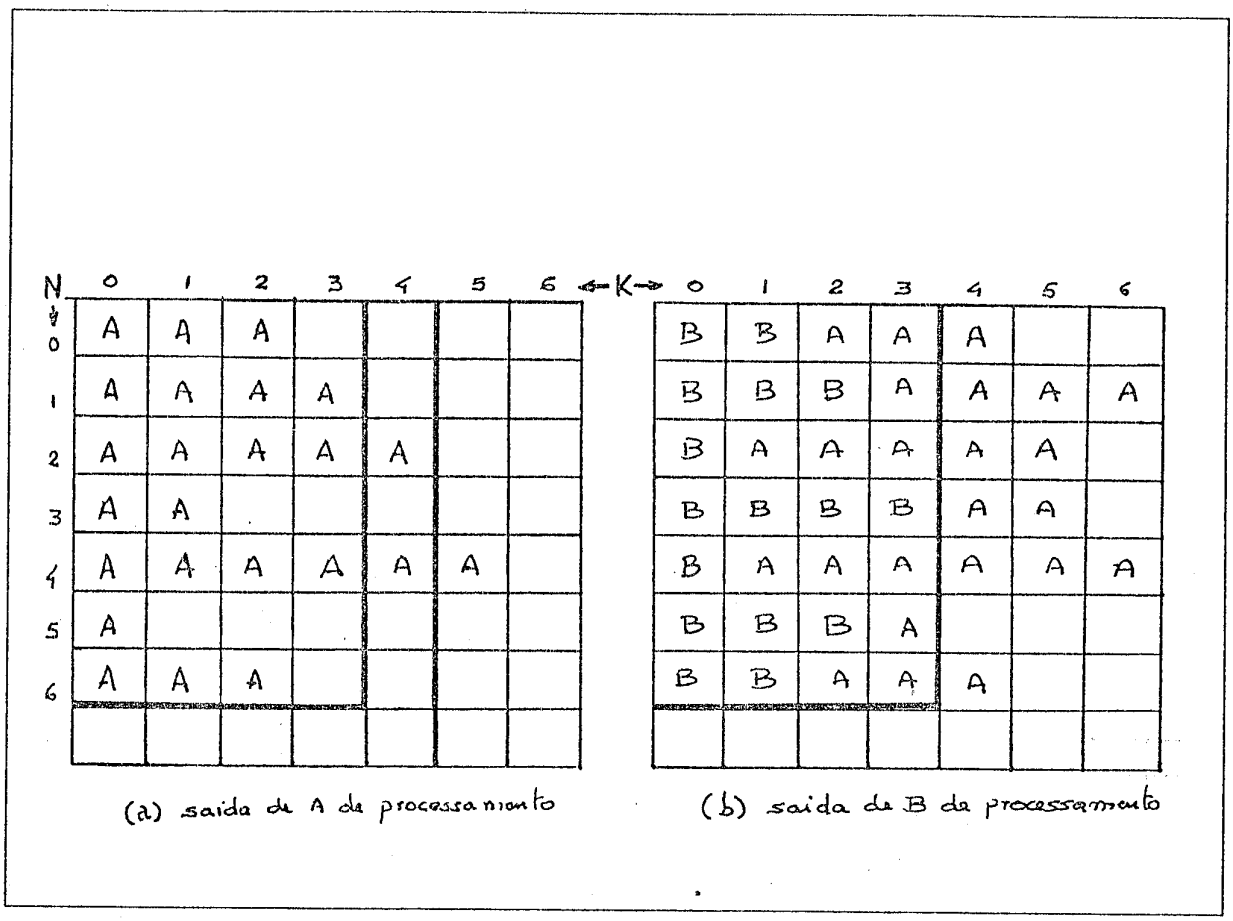


Figura 12 - Memória Cache e o conceito de "footprint"

A Figura 12(a) representa uma Memória Cache formada por sete conjuntos (N = 7), com um grau de associatividade de quatro (K = 4). Para melhor entender o conceito, vamos representar na Figura 12(a) um gráu de associatividade maior do que quatro e vamos considerar dois processos, A e B, competindo pela Memória Cache. A carga de uma linha na Memória Cache, por cada um dos processos, está representada pelas letras A e B, respectivamente a cada um dos processos correspondentes. Assim, a Figura 12(a) representa o "footprint" do Processo A na Memória Cache, no instante em que termina a sua fatia de tempo de processamento.

A Figura 12(b) representa o "footprint" do processo B no instante em que ele termina o seu processamento. Neste exemplo é possível observar que algumas linhas do processo A ainda se encontram na Memória Cache e poderão ser novamente utilizadas quando o processo A voltar ao processamento.

É importante observar que o "footprint" de um processo varia o seu tamanho ao longo do tempo, uma vez que, a medida em que as tarefas se sucedem, algumas linhas

Essa característica é especialmente observada na área de endereçamento destinada aos dados e faz com que seja impossível prever o comportamento da Memória Cache que tem um tamanho fixo.

Uma outra organização prevê a divisão da Memória Cache em duas partes, uma parte destinada ao uso do programa Supervisor e outra parte destinada ao uso do programa de usuários. Foi visto anteriormente que uma significativa parte dos "Cache Misses" ocorridos é devida ao chaveamento entre tarefas. Se o sistema operacional for preparado de tal forma a reiniciar o processamento de um mesmo programa após a ocorrência de uma interrupção, então a taxa de "Cache Misses" desse programa diminui sensivelmente. Por outro lado, se as mesmas interrupções ocorrerem com frequência, a taxa de "Cache Misses" devido a operações do supervisor tambem diminui.

Essa solução que sugere a divisão da Memória Cache em uma parte destinada ao supervisor e outra parte destinada ao programa dos usuários tem também sérios problemas. O primeiro deles está ligado ao fato de que é muito difícil prever se o sistema operacional está apto a devolver o controle para um mesmo programa, após a ocorrência de uma interrupção. Outro problema diz respeito ao fato de que a informação usada pelo supervisor é muitas vezes comum a informação usada pelo programa do usuário sendo permitido inclusive uma troca de informações entre o supervisor e o programa do usuário. Esse fato faz com que apareça com grande força o problema da consistência entre as duas partes da Memória Cache.

Finalmente, a terceira organização propõe a divisão da Memória Cache em níveis hierárquicos diferentes. Como também já foi visto, o tamanho da Memória Cache tem um papel significativo na taxa de "Cahe Misses". Em princípio, quanto maior for a Memória Cache, menor é a taxa de "Cache Misses" apresentada. O maior tamanho de Memória Cache conhecido, é da ordem de 128 Kilobytes, sendo o mais usual a utilização de Memórias Cache de 64 Kilobytes. Estas Memórias Cache de grandes dimensös apresentam dois graves problemas: o primeiro deles diz respeito ao tamanho físico e a complexidade dos seus circuitos lógicos o que aumenta sensivelmente o tempo de acesso aos dados. O segundo problema está relacionado ao custo de construção que é diretamente proporcional ao tamanho da Memória Cache.

Esses dois problemas fazem com que se pense em dividir a Memória Cache em partes distintas, segundo uma hierarquia de acesso previamente definida.

A soluções implementadas dentro dessa organização dividem a Memória Cache em dois níveis. O primeiro nível, correspondente a menor parte é geralmente na ordem de 4 Kilo-

remanescentes vão sendo reutilizadas. Mas, este tamanho é independente da organização da Memória Cache pois variarmos K ou N, o que ocorre é uma redistribuição diferente de dados de cada um dos processos, pelas linhas. Nesse caso, no entanto, vai haver uma variação na taxa de "Cache Misses".

Através do "footprint" é possível calcular a variação do número de "Cache Misses" em função do tamanho da Memória Cache (STON87). A Figura 13 mostra um exemplo dessa variação para uma Memória Cache utilizada no processamento de duas tarefas A e B, tendo a tarefa A um "footprint" de 1900 e a tarefa B, um "footprint" de 7900.

MEMÓRIAS CACHE COM ORGANIZAÇÃO ESPECIAL

Geralmente, os sistemas de computação que implementam o conceito de Memória Cache utilizam a organização descrita anteriormente ou seja, uma Memória Cache única construida em um único bloco. Isto se deve ao fato de ser esta implementação a mais simples uma vez que os componentes do processador tem somente que se referenciar a um único ponto de entrada ou saida seja para leitura de dados, escrita de dados ou chamada de instruções. No entanto, em alguns casos, é interessante estudar a utilização de organizações diferentes que, em certas circunstâncias apresentam várias vantagens.

Uma destas organizações propõe a divisão da Memória Cache em duas partes, uma para o armazenamento de instruções e outra para o armazenamento de dados. Essa organização apresenta uma grande vantagem em termos de desempenho pois atua diretamente na faixa de passagem e no tempo de acesso. A faixa de passagem é aumentada uma vez que a divisão da Memória Cache em dois blocos permite que cada um deles atenda separadamente pedidos vindos do processador. O tempo de acesso é melhorado uma vez que a divisão da Memória Cache em duas partes permite que cada uma seja colocada fisicamente mais próxima do circuito correspondente no processador. Existem, no entanto, alguns problemas tais como a consistência entre as duas partes e a otimização da utilização destas partes.

Em termos de consistência é necessário atentar para o fato de que a separação entre as instruções e os dados vai esbarrar no fato de que algumas instruções podem ser modificadas com base nos valores de determinadas variáveis e é óbvio que estas modificações devam sem refletidas na Memória Cache antes que a instrução seja executada.

Em termos de otimização da utilização da Memória Cache, é conhecido que a área de endereçamento ocupada por um programa tem o seu tamanho variável dinamicamente.

bytes tem um tempo de acesso extremamente pequeno e o segundo, de maior tamanho mas com tempo de acesso relativamente maior do que o tempo de acesso do primeiro nível. Embora a taxa de "Cache Misses" apresentada pelo primeiro nível seja alta, ela é compensada pela alta velocidade de acesso aos dados e, na média, o desempenho total da Memória Cache será maior.

IMPLEMENTAÇÕES DE MEMÓRIA CACHE

MODELO 85 DA LINHA /360 DA IBM

O modelo 85 da linha IBM /360 foi o primeiro sistema de computação comercial a implementar o conceito de hierarquia de memória (LIPT68). Embora o conceito não fosse à época novidade, essa implementação somente foi possível devido ao alcançe do nível de tecnologia necessário àquela implementação. A Figura 13 mostra a organização interna do Modelo 85 (CONT68).

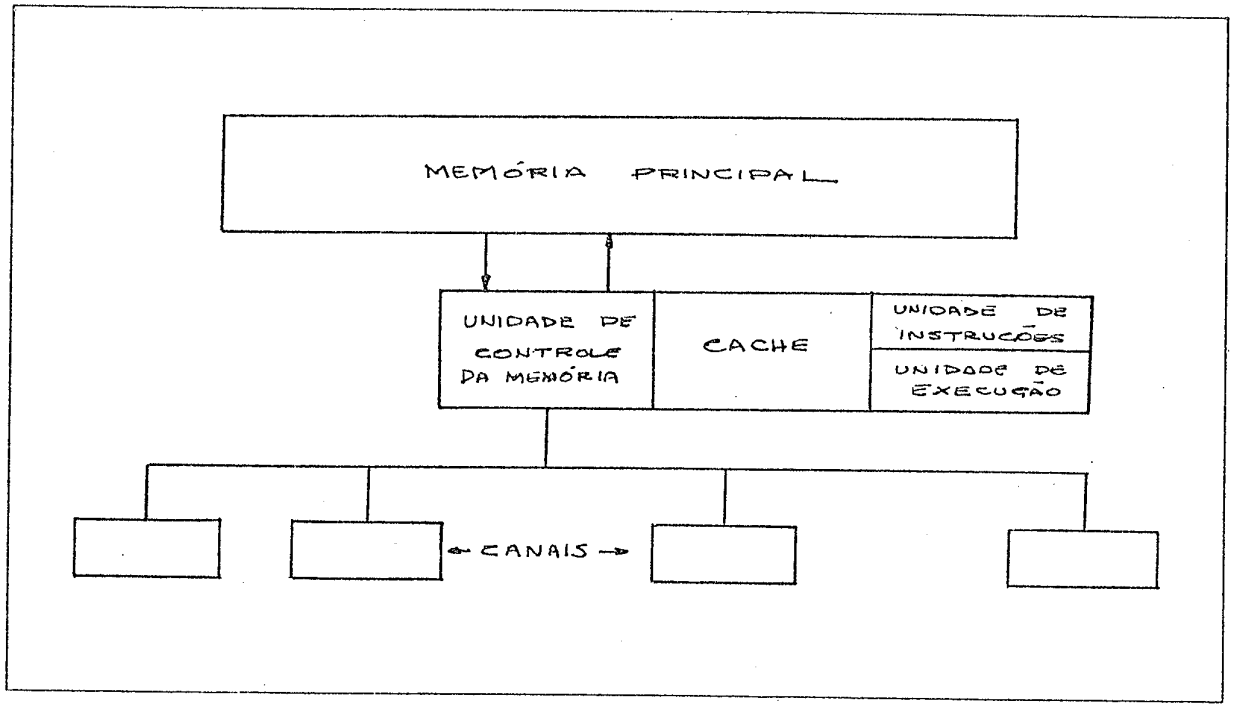


Figura 13 - Esquema funcional do modelo 85 da linha /360 da IBM

O modelo 85 tinha uma Memória Principal variando de 512 KBYTES até 4 MEGA-BYTES, com um ciclo de memória da ordem de 1,04 microsegundos. A Memória Cache era da ordem de 16 KBYTES podendo opcionalmente alcançar 24 KBYTES ou 32 KBYTES, com um tempo de ciclo de 80 nanosegundos. Tanto a Memória Principal como a Memória Cache do modelo 85 são logicamente divididas em setores cada um com a capacidade de 1 KBYTE conforme mostra a Figura 14.

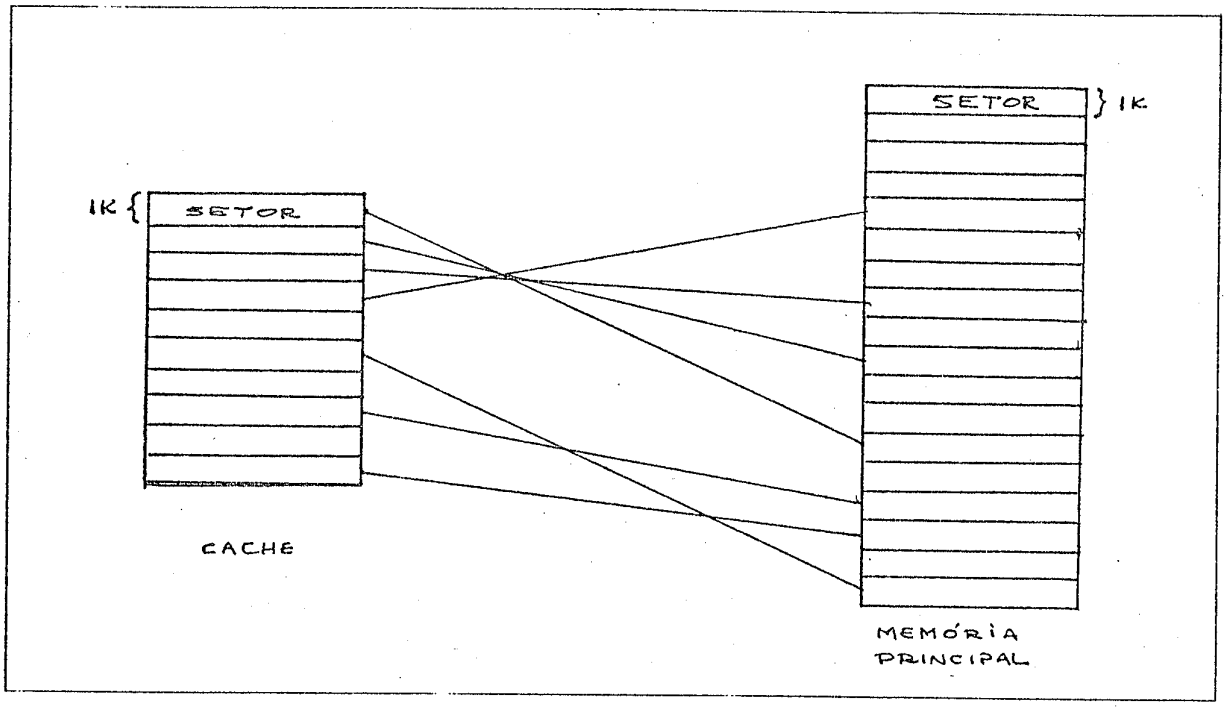


Figura 14 - Memória Cache do modelo 85 e o seu relacionamento com a Memória Principal

Cada setor da Memória Cache está associado a um setor da Memória Principal, sendo essa associação feita dinamicamente a medida em que dados de cada um dos setores são utilizados. Assim, somente estão relacionados setores cujos dados foram utilizados mais recentemente. O setor posicionado no topo da Memória Cache é aquele mais recentemente utilizado. A referência a um dado pertecente a um setor já posicionado na Memória Cache faz com que esse setor passe a ocupar o topo da Memória Cache. Quando a referência é feita a um dado que não está na Memória Cache, o setor correspondente a ele na Memória Principal é transferido para a Memória Cache indo ocupar o seu topo. Neste caso, o setor da Memória Cache posicionado no fundo da Memória Cache é então deletado para propiciar o lugar para aquele novo setor. Quando um setor da Memória Principal é referenciado, o seu conteudo total (1 KILOBYTE) não é totalmente copiado para a Memória Cache e sim um bloco, constituido de 64 bytes (cada setor portanto tem 16 blocos de 64 bytes cada um). Quando isso ocorre, somente um bloco é copiado para á Memória Cache. Se o setor é referenciado novamente, somento o bloco agora referenciado é então copiado a Memória Cache. Se for necessário, blocos sucessivos podem ser copiados em seqüência, mas sempre um de cada vez.

Se o processamento tem como resultado um armazenamento de dados na Memória Principal e se o dado que está sendo modificado está também na Memória Cache, este último também é atualizado. Em caso contrário nenhuma atividade ocorre na Memória Cache.

Dois ciclos de processamento são necessários para carregar um dado que está na Memória Cache. O primeiro ciclo é utilizado para determinar se o dado está na Memória Cache e o segundo ciclo é utilizado para ler este dado para o processador. Se o dado não está na Memória Cache, mais ciclos adicionais são necessários para trazê-lo até lá.

O modelo 85 opera internamente com uma palavra de 16 bytes e é esta a capacidade dos canais de dados internos. Assim, são necessários quatro ciclos de carga para copiar um bloco da Memória Principal para a Memória Cache. Para melhorar o desempenho, a referência a um dado na Memória Principal faz com que os 16 bytes que o contenham sejam primeiramente copiados para a Memória Cache, sendo então seguidos pelos três outros conjuntos de 16 bytes. Além disso, esse dado procurado é enviado diretamente para o processador com a finalidade de continuar com o processamento o mais rápido possível.

SISTEMA 3090 DA LINHA /370 DA IBM

O sistemaa linha IBM /370 é o sistema mais avançado dentro do grupo de grandes processadores da IBM. Dentro da arquitetura /370 este grupo teve início com o modelo IBM 3033, passando pela série IBM 308X.

Neste grupo, a tecnologia disponível permitiu que se partisse de um ciclo de processamento da ordem de 57 nanosegundos no IBM 3033 para 18.5 nanosegundos no IBM 3090. O sistema IBM 3090 usa o conceito de hierarquia de memória com a introdução, ao nível da Memória Principal, do conceito de expansão de memória o que permite um crescimento significativo da quantidade de memória disponível. a Figura 15 nos mostra um diagrama geral do modelo 400 do sistema IBM 3090.

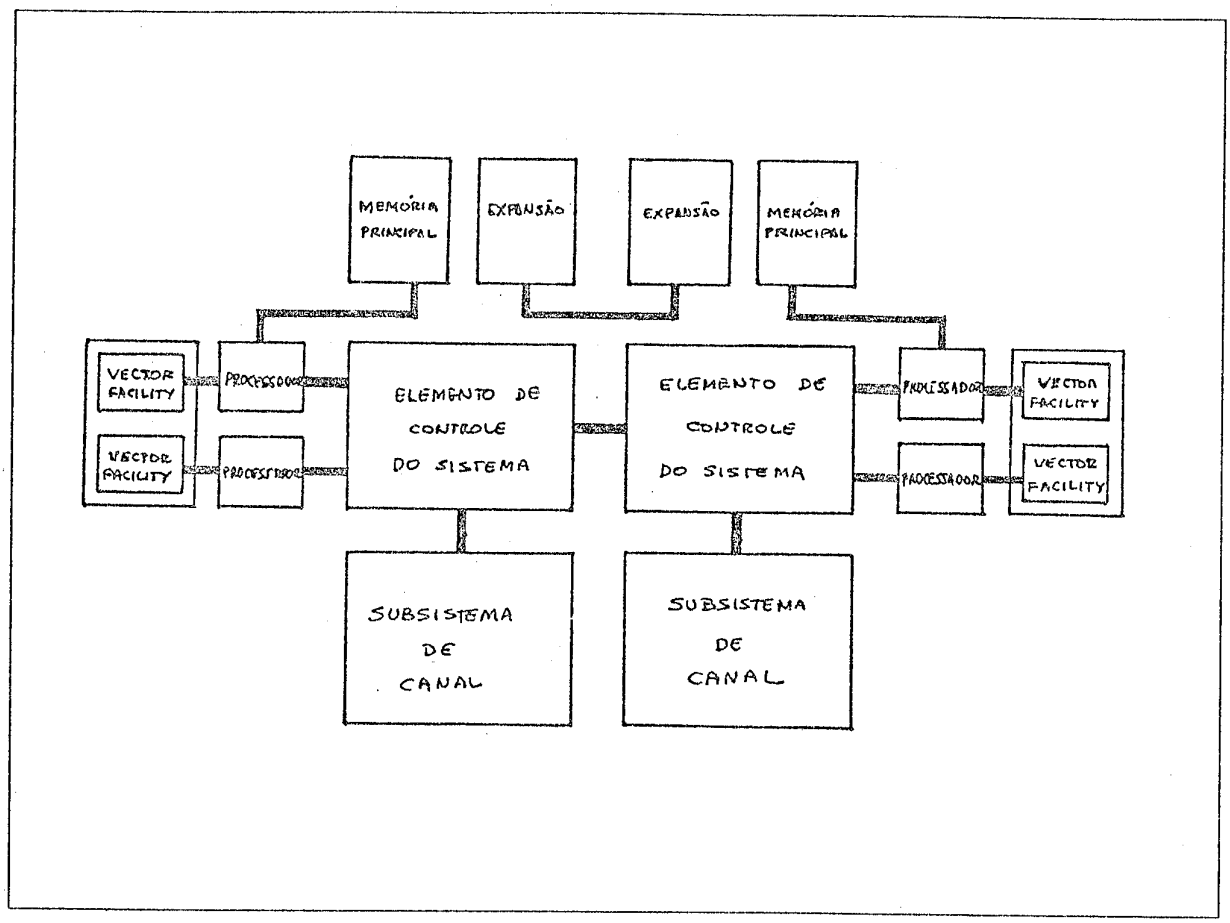


Figura 15 - Diagrama Geral do Modelo 400 do Sistema IBM 3090

O modelo 400 é um sistema de múltiplos processadores consistindo basicamente de dois modelos 200 interconectados por canais entre os seus elementos de controle e as suas expansões de memória. Os elementos de controle do sistema por sua vez interligam os processadores centrais com respectivos **vector facilities** aos canais.

A Memória Cache do sistema IBM 3090 tem uma capacidade de 64 KILOBYTES dividida em 8000 linhas de 8 bytes cada. Essas linhas estão divididas em quatro conjuntos tendo portanto cada conjunto, 2000 linhas. Esses 8 bytes correspondem na arquitetura do sistema a uma palavra dupla. Essa Memória Cache usa a filosofia "**store-in-cache**" onde uma atualização de dados resultante de um processamento vai primeiro para a Memória Cache e somente depois é então armazenada na Memória Principal. Uma linha da Memória Cache é mantida lá até que outro processador do sistema requisite aquela linha ou então até que haja necessidade de espaço naquela Memória Cache. Cabe ao Sistema de Controle gerenciar as diversas Memórias Cache de tal forma a saber onde está a versão mais atualizada daquela linha.

A cada linha da Memória Cache corresponde uma entrada no respectivo diretório. Cada diretório contém o endereço das linhas de suas Memória Cache e também é dividido nos mesmos quatro conjuntos. Em um acesso, as quatro entradas do diretório (uma de cada conjunto) são lidas simultaneamente e quatro linhas (palavras duplas) são acessadas.

Cada Memória Cache pode ser acessada nas seguintes maneiras: leitura para o processador, armazenamento do processador, transferência entre Memórias Cache e transferência da Memória Cache para a Memória Principal via os elementos de controle do sistema. Em todas essas modalidades a transferência é feita a uma taxa de uma palavra dupla (8 bytes) por ciclo. No entanto, a Memória Cache tem a possibilidade de ler ou escrever uma palavra quadupla por ciclo o que permite executar simultaneamente tipos diferentes de acesso. Como exemplo é possível que uma transferência de dados para a Memória Principal seja feita simultaneamente com um acesso do processador.

MODELO C120 DA CONVEX

Na linha de supercomputadores a CONVEX apresenta dois modelos; o modelo C-120 e a série C-200 onde cada um dos modelos seguem arquiteturas diferentes. No modelo C-120, mostrado na Figura 16, o tempo de acesso a Memória Cache é da ordem de 100 nanosegundos. Este modelo apresenta uma Memória Cache de 64 KILOBYTES que tem como característica a possibilidade de ser bloqueada quando da necessidade de um acesso direto a Memória Principal. Além disso, existem duas Memórias Cache secundárias, uma para a unidade de tradução de endereços e outra para o processador propriamente dito. Essas Memórias Cache secundárias, de menor capacidade mas fortemente acopladas as suas unidades atendem a referências imediatas e de maior frequência.

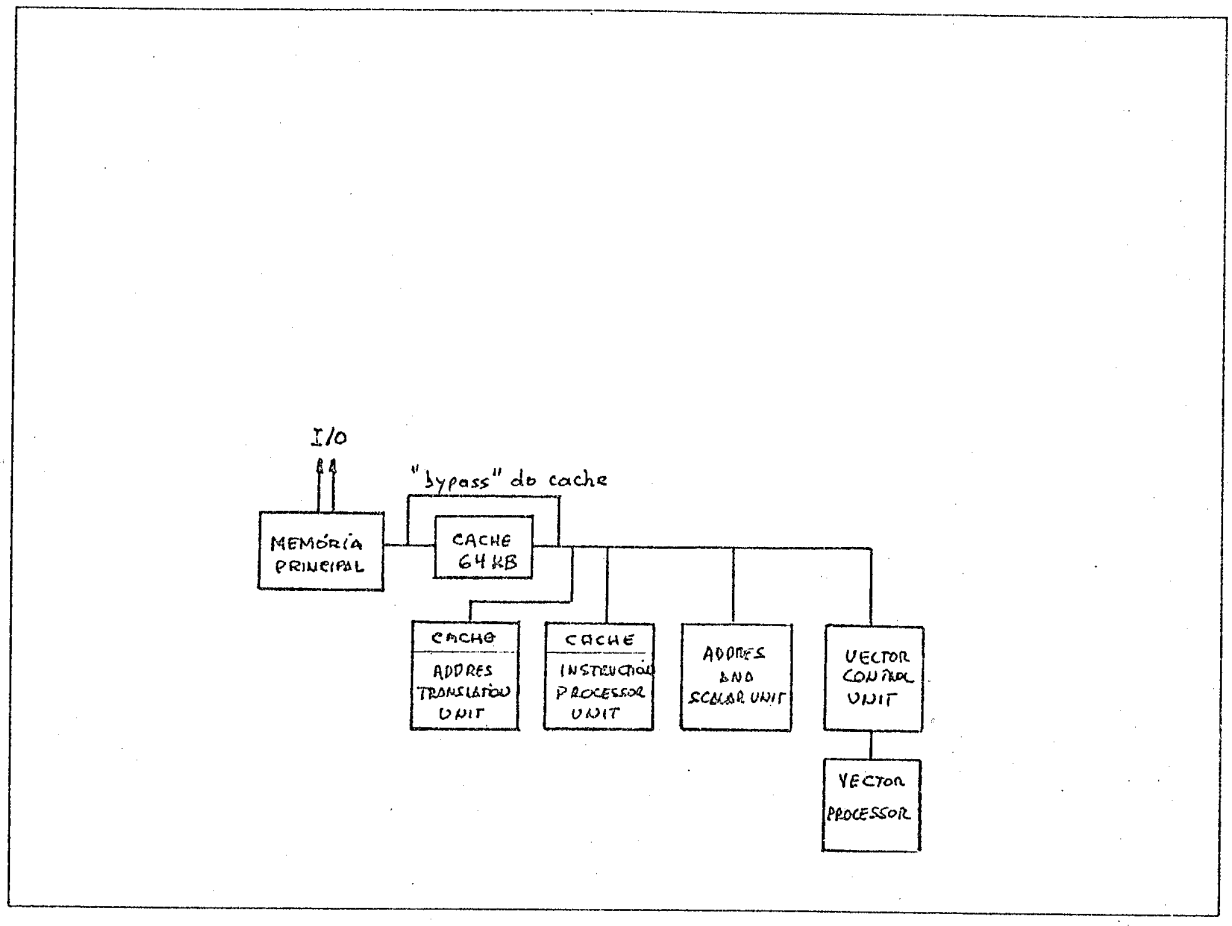


Figura 16 - Diagrama em Blocos do Sistema C-120 da CONVEX

MODELO SX-2 DA NEC

O Modelo SX-2 da NEC é um Supercomputador que apresenta processadores separados para cada função. Esse modelo possui quatro processadores de aplicação divididos cada um em duas unidades funcionais ou sejam unidade vetorial e unidade escalar. A Memória Cache de cada um desses processadores está localizada na unidade escalar e tem acesso direto a Memória Principal. Sua capacidade é de 64 KILOBYTES e está organizada em um conjunto de linha com capacidade de 8 bytes por linha. A arquitetura do modelo SX-3 da NEC está mostrada na Figura 17.

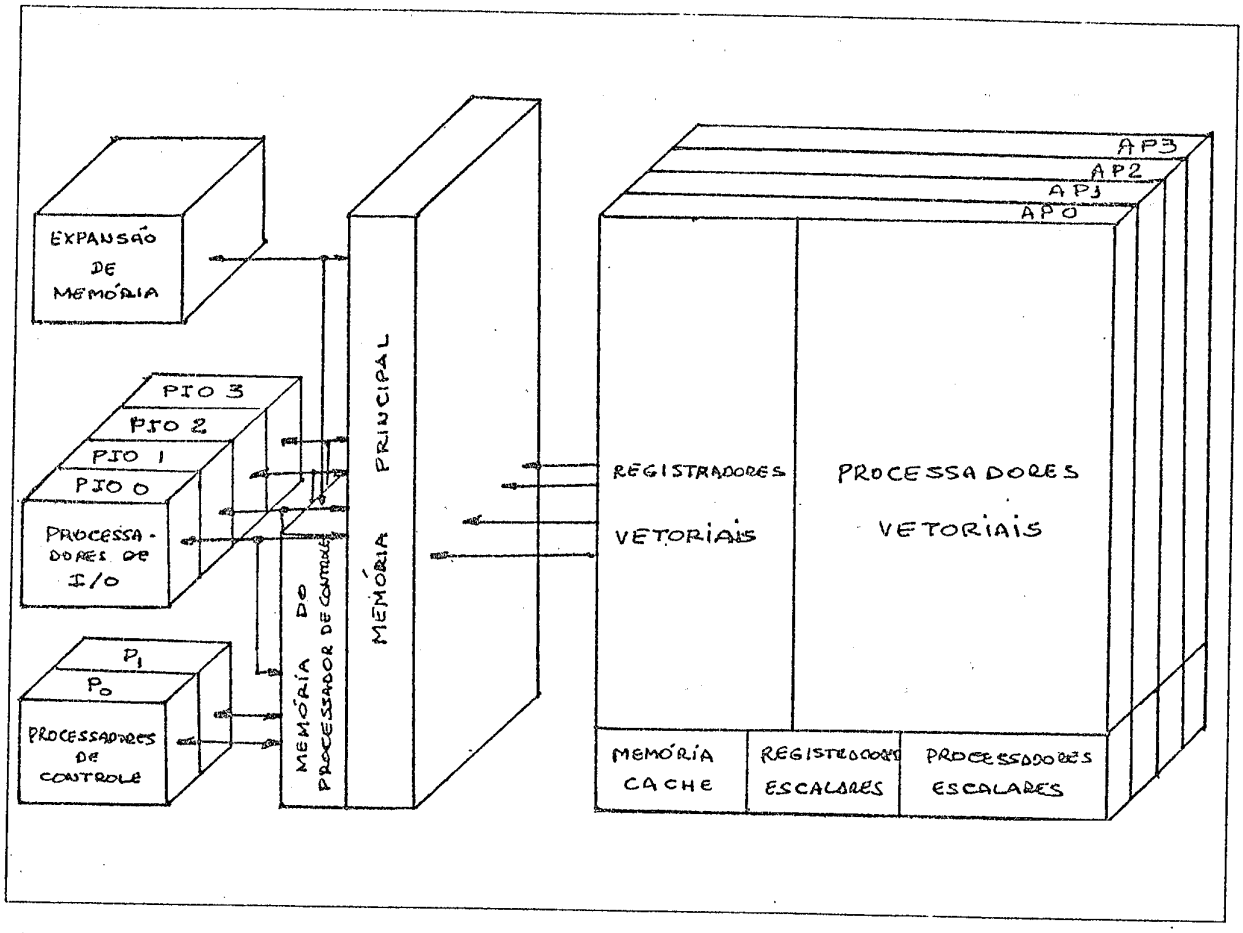


Figura 17 - Arquitetura do modelo SX-3 da NEC

BIBLIOGRAFIA

COHE89 Cohen E.I., King G.M. & Brady J.T.
Storage Hierarchies
IBM Systems Journal, Vol 28 N 1, 1989

CONT68 Conti, C.J. Gibson, D.H. & Pitkowsky, S.H.
Structural Aspects of the System/360 Model 85
IBM Systems Journal, Vol 7 N 1, 1968

GIND77 Gindele, J.D.
Buffer Block Prefetching Method
IBM Technical Disclosure Bulletin, 20, 2, July 1977

PERR89 Perry T.S.
Intel's Secret Is Out
IEEE Spectrum, April 1989

SMIT82 Smith, A.J.
Cache Memories
Computing Surveys, Vol 14 N 3, September 1982

SMIT87 Smith, A.J.
Cache Memory Design: An Evolving Art
IEEE Spectrum, December 1987

STON87 Stone H.S.
High Performance Computer Architecture
Addison Wesley, 1987

TUCK86 Tucker, S.G.
The IBM 3090 System: An Overview
IBM Systems Journal, Vol 25, N 1 1986