Avaliação 1 - Disciplina: CMP1059/C01

Goiânia: 29/09/2025 Valor: 10,0

Nome: Myllena Rodrigues Oliveira Nota:

A avaliação é sem consulta e individual. A consulta faz parte da avaliação. Questões rasuradas não serão corrigidas.

Marque V para as alternativas verdadeiras e F para as alternativas falsas.

Memória Cache

(V) Em uma hierarquia de memória típica, conforme descemos dos registradores para caches (L1, L2, L3), depois DRAM e por fim armazenamento externo, o custo por bit diminui, a capacidade aumenta e o tempo de acesso cresce; o objetivo do projeto é manter, o máximo possível, os dados e instruções frequentemente usados nos níveis mais rápidos para reduzir o tempo médio de acesso do processador.

(V) O princípio da localidade de referência explica por que caches funcionam: programas tendem a acessar repetidamente um pequeno conjunto de instruções e dados por curtos períodos, como laços e sub-rotinas, e estruturas contíguas como tabelas e arrays; por isso, trazer um bloco para a cache tende a aproveitar acessos subsequentes próximos no espaço e no tempo.

(V) Em uma organização de dois níveis, se a fração de acertos H na cache mais rápida é alta, o tempo médio de acesso aproxima-se do tempo da cache, pois apenas a fração (1-H) paga a penalidade de buscar o bloco na memória inferior e então entregá-lo ao processador.

(V) A unidade de transferência para memória principal e cache é o “bloco” (ou “linha” na cache), que contém várias palavras/bytes; assim, mesmo que o processador solicite uma palavra, a cache transfere e armazena um bloco inteiro, porque isso aumenta a chance de futuros acertos pelo efeito da localidade.

(V) No mapeamento direto, cada bloco da memória principal só pode ocupar uma única linha específica da cache, determinada por uma função simples (ex.: i = j mod m); isso simplifica o hardware, mas pode causar thrashing se dois blocos frequentemente acessados competirem pela mesma linha.

(V) No mapeamento totalmente associativo, um bloco pode ser carregado em qualquer linha da cache; a decisão de presença é feita comparando a tag do bloco com as tags de

todas as linhas em paralelo, o que aumenta a flexibilidade e a taxa de acertos, mas exige comparadores em massa e controle mais complexo.

(V) O mapeamento associativo em conjunto (set-associative) é um meio-termo: a cache é dividida em conjuntos e cada conjunto tem k linhas; um bloco mapeia para um conjunto específico (por índice), mas pode ocupar ualquer uma das k linhas daquele conjunto, reduzindo conflitos em relação ao mapeamento direto.

(V) Experimentos de simulação indicam que passar de direto para 2-vias melhora significativamente a razão de acertos até cerca de 64 KB, enquanto aumentar de 2 para 4 vias traz ganho menor; além de 32 KB, crescer ainda mais a cache tende a produzir ganhos marginais, dependendo da carga de trabalho.

(V) Em caches set-associative de 2 vias, o algoritmo LRU pode ser implementado de forma simples com um bit de uso por linha do conjunto; a linha com USE=0 é substituída, refletindo a suposição de que o bloco menos recentemente usado é o menos provável de ser referenciado em breve.

(V) Substituição aleatória costuma ter desempenho apenas ligeiramente inferior aos algoritmos baseados em uso, como LRU, conforme estudos de simulação citados; embora simples, a escolha aleatória perde pouco em relação às escolhas “informadas” pelo histórico de acessos.

(V) Em write-through, toda escrita atualiza simultaneamente cache e memória principal, simplificando a consistência com DMA/IO e entre processadores, porém aumentando o tráfego no barramento; em write-back, as escritas ficam retidas na cache e vão à memória apenas quando a linha é substituída, reduzindo tráfego, mas complicando coerência.

(V) Em cargas típicas, algo como 15% das referências são escritas, mas em HPC esse percentual pode subir (aprox 33% em multiplicação de vetores e até aprox 50% em transposição de matrizes), o que influencia a escolha da política de escrita e o dimensionamento do subsistema de memória.

(V) Uma victim cache pequena, totalmente associativa (tipicamente 4 a 16 linhas), inserida entre a L1 mapeada diretamente e o próximo nível, pode reduzir perdas por conflito ao “reciclar” blocos recém-evictados que têm alta probabilidade de serem referenciados novamente logo após a substituição.

(V) A operação de leitura típica consulta a cache primeiro; em acerto, os buffers para o barramento permanecem inativos e a transferência ocorre apenas entre CPU e cache; em falha, o endereço é colocado no barramento, o bloco é trazido da memória principal para a cache e então a palavra é entregue à CPU.

(V) Em caches físicas, os dados são indexados por endereços físicos após a tradução da MMU; em caches lógicas (virtuais), a cache é consultada com endereços virtuais antes da MMU, o que pode reduzir latência, mas exige esvaziamento a cada troca de contexto ou tags ampliadas para distinguir espaços de endereços de processos.

(V) O endereço da memória pode ser decomposto em campos Tag, Linha/Conjunto e Palavra/Deslocamento; o tamanho desses campos depende da função de mapeamento e determina tanto quantas linhas ou conjuntos existem quanto quantos bits de tag são necessários em cada entrada da cache.

(F) Em mapeamento direto, dois blocos diferentes da memória principal nunca mapeiam para a mesma linha da cache, o que elimina conflitos de posição e torna impossível o thrashing dentro de uma linha.

(F) No mapeamento totalmente associativo, o número de linhas da cache é determinado rigidamente pelos bits de endereço, porque parte do endereço é usada para escolher a linha específica antes da comparação de tags.

(V) A taxa de transferência para memória de acesso aleatório é dada por 1/(tempo de ciclo), enquanto para memórias de acesso não aleatório vale a relação TN = TA + n/R, onde TA é o tempo de acesso, n o número de bits e R a taxa em bps; isso quantifica o custo de posicionamento e de transferência sustentada.

(V) O tamanho da linha influencia o aproveitamento da localidade espacial: linhas maiores aumentam a chance de acerto em leituras sequenciais, mas excessos podem trazer desperdício (dados não usados) e aumentar latência de falha; o projeto precisa equilibrar esses efeitos conforme a carga de trabalho.

(V) Em sistemas com múltiplos processadores e caches próprias, é necessário um protocolo ou técnica de coerência de cache, pois uma escrita em uma cache pode invalidar cópias de outras caches; soluções incluem writethrough com observação de barramento, transparência por hardware ou marcar regiões como não-cacheáveis.

(V) A cache não costuma ser visível ao programador: ela é um mecanismo de hardware que organiza a movimentação de dados entre memória principal e registradores para melhorar desempenho, sem exigir instruções explícitas de software para sua gestão no fluxo normal.

(V) A escolha do tamanho da cache envolve restrições de área no chip/placa e impacto de latência: caches maiores tendem a ser levemente mais lentas que caches menores, mesmo na mesma tecnologia e posição física, de modo que não existe “tamanho ótimo” universal para todas as cargas.

(V) Em sistemas com memória virtual, posicionar a cache antes da MMU (cache virtual) pode reduzir latência, porém requer mecanismos adicionais para distinguir contextos de processos ou limpeza em troca de contexto; já a cache física evita esse problema ao operar com endereços traduzidos.

(F) Em write-back, as atualizações são imediatamente refletidas na memória principal a cada escrita, o que minimiza o tráfego no barramento e elimina a necessidade de bits de modificação por linha.

(V) A política FIFO substitui a linha que está há mais tempo no conjunto independentemente do uso recente; é simples de implementar (p. ex., round-robin), mas pode ser inferior a LRU quando a localidade temporal é forte, pois ignora a recência de referência.

(V) Em LFU, cada linha mantém um contador de referências, e a linha menos frequentemente usada é escolhida para substituição; embora intuitivo, o custo de manter contadores pode ser maior e nem sempre captura bem "fases" curtas de acesso, nas quais LRU costuma se sair melhor.

(V) O fluxo de uma falha de leitura envolve selecionar uma linha no conjunto (pelo algoritmo de substituição), copiar o bloco da memória principal para a cache e só então disponibilizar a palavra à CPU; em algumas organizações, a transferência para a cache e para a CPU ocorrem em sequência após o preenchimento da linha.

(V) Mesmo sem alterar a tecnologia de memória, o uso de múltiplos níveis de cache (L1/L2/L3) permite aproximar o tempo médio de acesso do tempo do nível mais rápido, desde que a localidade garanta altas razões de acerto nos níveis superiores e que o custo de falhas seja amortizado pelos acertos.

(V) Uma parte da memória principal pode ser usada como "cache de disco" via software, agrupando escritas e servindo leituras recentes rapidamente, reduzindo acessos pequenos e frequentes ao disco; embora fora do escopo do capítulo para detalhes, o princípio amplia a hierarquia lógica de armazenamento.

(F) Uma cache write-back atualiza imediatamente a memória principal a cada escrita, dispensando o uso de qualquer bit de modificação ("dirty") e evitando gargalos de coerência em sistemas com múltiplas caches.

(F) No mapeamento associativo total, parte do endereço serve para escolher previamente uma linha específica da cache; assim, o número de linhas fica determinado pelo formato do endereço, reduzindo a necessidade de comparadores paralelos.

(F) A victim cache é tipicamente uma pequena cache mapeada diretamente, inserida após a L2, cuja função principal é aumentar a taxa de write-through para reduzir tráfego de barramento em sistemas DMA.

(F) A hierarquia de memória ideal utiliza apenas a memória mais veloz em todos os níveis, já que custo/bit e capacidade se mantêm constantes conforme descemos para DRAM e armazenamento externo.

(F) Em caches set-associative, um bloco da memória principal pode mapear para qualquer conjunto, pois o índice do conjunto é escolhido dinamicamente em tempo de execução conforme o histórico de acessos.

(F) Com write-through, as outras caches permanecem sempre coerentes sem medidas adicionais, já que cada escrita atualiza a memória e invalida automaticamente qualquer cópia remota.

(F) O fluxo de leitura típico envia todo acesso diretamente ao barramento do sistema; a cache é consultada apenas após a DRAM responder, para confirmar se houve acerto.

(F) O tamanho da linha de cache inclui apenas dados e nunca contém bits de controle ou tag, pois estes ficam armazenados na memória principal junto ao bloco original.

(F) Em uma cache write-back, não existe cenário em que write-through seja mais eficiente, já que write-back sempre evita escritas adicionais independentemente da frequência de atualizações por linha.

(F) Em sistemas multiprocessados, manter a coerência de cache dispensa protocolos; basta marcar regiões compartilhadas como cacheáveis e confiar no tempo médio de acesso.

(F) No mapeamento direto, não há risco de conflito porque cada bloco da memória principal pode ocupar qualquer linha livre; o controlador escolhe a linha menos usada.

(F) O conceito de localidade de referência é marginal para o projeto de caches, pois a principal vantagem vem de acessos totalmente aleatórios que exploram paralelismo de barramento.

(F) O número de linhas da cache, em mapeamento associativo total, é fixado pelo campo "número de linha" presente no endereço e, por isso, cresce com o espaço de endereçamento.

(F) Para reduzir conflitos, a melhor prática é diminuir o tamanho da linha ao mínimo absoluto, já que linhas maiores sempre degradam a taxa de acertos ao trazerem dados inúteis.

(F) Em write-back, dispositivos de E/S podem acessar livremente a memória principal sem coordenação com a cache, pois ela nunca mantém cópias mais novas que a DRAM.

(F) A decomposição do endereço em Tag/Índice/Palavra é irrelevante para o projeto; a cache identifica blocos apenas pelo número do barramento gerado pelo processador.

(F) O aumento do tamanho total da cache não tem qualquer impacto na sua latência; caches maiores são sempre tão rápidas quanto caches menores na mesma tecnologia.

(F) A política de substituição FIFO considera a recência de uso para retirar o bloco menos recentemente utilizado, sendo funcionalmente idêntica a LRU em todas as cargas.

(F) Em set-associative, a implementação física não pode ser vista como múltiplas caches diretas; por definição, cada via é sempre associativa total.

(F) A cache costuma ser gerenciada por software de aplicação, com instruções explícitas para mover blocos entre DRAM e cache a cada acesso do programa.

(F) Em cargas típicas, praticamente todas as referências são escritas (aprox 80–90%), razão pela qual writethrough é sempre inviável por excesso de tráfego.

(F) A transferência em caso de hit precisa sempre passar pelo barramento do sistema antes de chegar à CPU, garantindo consistência com a DRAM a cada leitura.

(F) O termo “linha de cache” refere-se exclusivamente ao bloco de DRAM, já que tag e bits de controle pertencem ao controlador externo e não ocupam espaço na cache.

(F) No associativo em conjunto, aumentar k (número de vias) não altera o mapeamento: o bloco continua restrito a uma única posição fixa dentro do conjunto.

(F) Uma victim cache amplia o tempo de acesso crítico da L1, pois precisa ser consultada em série antes de todo hit normal na L1, tornando o caminho comum mais lento.

(F) Em leitura, primeiro atualiza-se a DRAM com o bloco pedido para só depois copiá-lo à cache, garantindo que a memória principal mantenha sempre a cópia mais recente.

Introdução aos processadores

(V) A organização do processador reúne ALU, unidade de controle e um conjunto de registradores internos conectados por um barramento interno, permitindo mover instruções e dados entre PC, IR, MAR e MBR até a ALU, que só opera sobre dados presentes nesses registradores.

(V) Registradores visíveis ao usuário reduzem acessos à memória ao reter operandos temporários; podem ser de uso geral ou especializados (dados, endereços, índices, ponteiros de pilha/segmento), com maior flexibilidade quando ortogonais, mas com economia de bits de opcode quando há especialização.

(V) Os registradores de controle/estado governam a operação: PC aponta a próxima instrução; IR guarda a instrução lida; MAR e MBR mediam endereços e palavras trocadas com a memória; a PSW agrega flags (zero, sinal, carry, overflow), controle de interrupções e modo supervisor/usuário.

(V) Em uma chamada de sub-rotina, o salvamento de registradores visíveis pode ser automático por hardware em algumas arquiteturas, ou responsabilidade do programa em outras; o objetivo é restaurar o contexto no retorno sem interferir no estado do chamador.

(V) O tamanho e a quantidade de registradores impactam codificação e tráfego de memória: 8–32 registradores costuma ser um intervalo eficaz; poucos elevam cargas/armazenamentos, muitos exigem mais bits nos especificadores e tendem a reduzir marginalmente acessos adicionais.

(V) O PC normalmente é incrementado após a busca, de modo que a próxima instrução esteja endereçada; desvios e saltos alteram o PC explicitamente, redirecionando o fluxo.

(V) O ciclo indireto trata operandos cujo endereço efetivo requer uma referência intermediária: após identificar indireção no IR, transfere-se o campo de referência ao MAR, realiza-se leitura e carrega-se no MBR o endereço efetivo para uso na execução.

(V) Um pipeline de instruções pode ser decomposto em FI, DI, CO, FO, EI e WO; com estágios balanceados, processa múltiplas instruções simultaneamente, reduzindo o tempo médio por instrução quando o pipeline está cheio e sem conflitos.

(V) Desvios condicionais introduzem penalidade de esvaziamento quando a predição falha; técnicas como busca do alvo, buffer de laço, múltiplos fluxos e predição (estática ou dinâmica com tabela de histórico e 2 bits saturantes) mitigam as bolhas.

(V) Hazards estruturais surgem quando dois estágios competem por um recurso único (p. ex., memória de uma porta); a mitigação envolve duplicar recursos (multiplas portas/unidades) ou escalonar para evitar contendas.

(V) Hazards de dados incluem RAW (verdadeira dependência), WAR (antidependência) e WAW (dependência de saída); sem encaminhamento e/ou reordenação apropriados, o pipeline precisa inserir bolhas para preservar a semântica sequencial.

(V) Um exemplo clássico de RAW: uma ADD escreve EAX tardiamente e a SUB seguinte precisa de EAX mais cedo; sem bypass, é necessário retardar a SUB até o valor correto estar disponível no estágio de leitura/execução apropriado.

(V) O ganho ideal teórico de um pipeline com k estágios e n instruções sem desvios tende a k quando $n \rightarrow \infty$, com tempo total aprox [k+(n-1)] * t; na prática, desequilíbrios de estágio, d (overhead entre estágios) e falhas de predição reduzem o speedup.

(V) O modo supervisor permite executar instruções privilegiadas e acessar regiões protegidas; um bit de modo na PSW distingue supervisor de usuário, reforçando proteção e isolamento.

(V) Em arquiteturas com segmentação, registradores de segmento guardam bases de segmentos distintos (código, dados, pilha), e registradores de índice/pilha suportam modos de endereçamento e operações de pilha com endereçamento implícito.

(V) Um barramento interno do processador integra registradores, ALU e unidade de controle, viabilizando transferências internas sem ocupar o barramento do sistema; MAR liga-se ao barramento de endereços e MBR ao de dados para trocas com a memória.

(V) Durante interrupções habilitadas, o conteúdo do PC é salvo (tipicamente via MBR em área reservada ou pilha), e o PC recebe o endereço da rotina de serviço; ao término, o estado salvo restaura o fluxo interrompido.

(V) O uso de registradores dedicados (como acumulador implícito, ponteiros e segmentos) compacta o código mas pode reduzir flexibilidade; já a ortogonalidade maximiza liberdade de mapeamento de operandos ao custo de mais bits de codificação.

(V) Buffers entre estágios de pipeline são essenciais para segurar resultados intermediários; porém acrescentam atraso d de transferência e controle, que aumenta o tempo de ciclo e limita ganhos quando os estágios já estão curtos.

(V) A análise de hazards estruturais pode ser auxiliada por “tabelas de reservas”, que identificam contendas temporais de recursos por estágio, orientando duplicação de unidades ou reescalonamento.

(F) O IR contém sempre os dados do operando fonte principal, dispensando MBR e leitura de memória, já que a instrução embute tanto a operação quanto os dados completos de entrada.

(F) Em todas as arquiteturas, o salvamento de registradores na chamada de sub-rotina é automático por hardware; programadores jamais precisam inserir prólogo/epílogo para preservar contexto.

(F) Como regra, quanto mais registradores, melhor: duplicar de 32 para 64 reduz sempre pela metade os acessos à memória, sem impactos em tamanho de opcode ou complexidade.

(F) O PC jamais é alterado por instruções de desvio; o redirecionamento do fluxo é feito por um registrador separado que não interfere na sequência de busca.

(F) O ciclo indireto não envolve leituras adicionais: o endereço efetivo é sempre conhecido após a decodificação, independentemente do modo de endereçamento especificado.

(F) Em pipeline, prefetch elimina integralmente a penalidade de desvios, tornando desnecessárias técnicas como buffer de laço, predição e múltiplos fluxos.

(F) Hazards estruturais não existem se a memória principal é única; a serialização natural dos acessos já garante paralelismo entre busca e acesso a dados.

(F) Hazards de dados do tipo WAR e WAW não podem ocorrer em pipelines; somente RAW é possível, pois as escritas sempre acontecem antes de qualquer leitura subsequente.

(F) A penalidade de desvio desaparece aumentando arbitrariamente o número de estágios; quanto mais profundo, menor o custo de errar a previsão.

(F) O modo supervisor não influencia acesso à memória; qualquer instrução pode tocar regiões protegidas se endereçar corretamente, sem necessidade de privilégios.

(F) Registradores de segmento e de pilha são redundantes em arquiteturas com índices; qualquer registrador de uso geral substitui suas funções sem implicações na codificação.

(F) MAR e MBR são decorativos: a ALU comunica-se diretamente com a memória principal a cada operação, e a presença de registradores intermediários não altera o fluxo.

(F) Durante uma interrupção, o processador continua executando a instrução seguinte e apenas registra em log o pedido; quando “sobrar tempo”, atende a rotina.

(F) Especializar registradores nunca reduz o tamanho do opcode; sempre aumenta o número de bits necessários, pois é preciso identificar a classe e o índice do registrador.

(F) Buffers entre estágios só melhoram o throughput e não impõem nenhum custo temporal adicional; com eles, o tempo de ciclo sempre diminui.

(F) A previsão com 2 bits muda de estado com um único erro, alternando imediatamente entre “tomado” e “não tomado”, o que evita persistência em decisões ruins.

(F) O cálculo de endereço efetivo (CO) nunca depende de resultados de instruções anteriores; portanto, não participa de hazards de dados.

(F) O ganho de desempenho de um pipeline cresce indefinidamente com n, ultrapassando o número de estágios k quando o programa é grande.

(F) Tabelas de reservas são usadas apenas para prever instruções de desvio; não têm relação com análise de recursos como ALUs ou portas de memória.

(F) A PSW armazena exclusivamente resultados aritméticos (zero/sinal) e nunca contém bits de controle como habilitar interrupções ou modo supervisor.

(F) Índices e ponteiros de pilha não influenciam modos de endereçamento; servem somente como registradores de dados comuns sem semântica implícita.

(F) Registradores compõem um grande banco de armazenamento geral, com milhares de posições numeradas como a memória principal; por isso, a movimentação de dados entre DRAM e CPU ocorre diretamente, sem intermediação de níveis mais rápidos. Na prática, seu papel é aproximar a velocidade de dados à do núcleo, reduzindo esperas sobre instruções e operandos.

(V) Em uma hierarquia de memória, os registradores ocupam o topo por serem o tipo mais rápido e também o menor e mais caro, servindo como o repositório imediato que alimenta a execução de instruções. Essa posição permite que a maior parte do trabalho recente do processador ocorra sem recorrer a níveis mais lentos.

(F) É comum que o número de registradores disponíveis para a CPU seja da mesma ordem de grandeza da memória cache L1, permitindo mapear quase um para um as palavras ativas do programa sem deslocamentos adicionais. Essa equivalência simplifica a gerência de dados em tempo de execução.

(V) Embora a quantidade de registradores varie entre projetos, é comum encontrar algumas dezenas, existindo arquiteturas com contagens maiores que chegam a centenas. Essa variação busca equilibrar área, energia e latência de acesso, sem perder o benefício de manter operandos imediatos perto da unidade de execução.

(F) Registradores são normalmente implementados com tecnologias não voláteis semelhantes a discos magnéticos, retendo o conteúdo entre desligamentos para acelerar reinicializações e retomadas de programas. Assim, servem como um “estado quente” persistente da CPU.

(V) O objetivo prático dos registradores é reduzir o tempo de espera do processador por instruções e operandos, aproximando a taxa de entrega de dados da cadência do núcleo.

Ao manter valores de uso imediato, eles evitam acessos frequentes a níveis mais lentos da hierarquia.

(F) A transferência entre a memória principal e os registradores ocorre, por padrão, sem a participação de caches, que são ignoradas para reduzir latências. Essa rota direta minimiza o tráfego total no sistema.

(V) Como o custo por bit cresce conforme se busca menor latência, o projeto usa poucos registradores extremamente rápidos e muito mais memória lenta nos níveis inferiores. Essa assimetria é um compromisso entre velocidade e economia.

(F) Por terem velocidade superior, registradores dispensam o uso de cache, já que qualquer acesso à DRAM pode ser atendido no mesmo ritmo, tornando redundante a presença de níveis intermediários.

(V) A posição dos registradores no topo implica que a maioria dos dados “quentes” de curtíssimo prazo deve estar ali quando a execução é eficiente, enquanto conjuntos maiores e menos imediatos residem temporariamente em cache e, por fim, na DRAM.

(F) Em termos de interface, registradores são endereçados como locais da memória principal, possuindo endereços exclusivos no mesmo espaço de endereçamento usado pelas DRAMs, o que simplifica o hardware de busca.

(V) Entre as memórias internas (registradores, caches, DRAM), todas são descritas como normalmente voláteis e implementadas com tecnologia semicondutora, reforçando que seu conteúdo não sobrevive ao desligamento.

(F) O ganho de desempenho proporcionado pelos registradores independe de localidade de referência, pois eles atuam apenas como cópias permanentes de toda a imagem do programa em execução.

(V) Como o processador tende a operar, por curtos intervalos, sobre pequenos conjuntos de instruções e dados, os registradores são ideais para reter os valores mais críticos de ciclo a ciclo, explorando intensamente a localidade temporal.

(F) O número de registradores em um projeto contemporâneo é ditado diretamente pela capacidade total da cache L2, uma vez que ambas as estruturas derivam dos mesmos campos de endereço de memória.

(V) A hierarquia eficaz requer que os registradores absorvam a maior parte das operações imediatas do núcleo, com a cache retendo subconjuntos recentes da DRAM; assim, acessos a níveis lentos são amortizados, e a execução tende a fluir sem paradas frequentes.

(F) Por definição, registradores pertencem à memória externa do sistema e são acessados como arquivos e registros, de modo semelhante a discos e fitas, com persistência entre execuções.

(V) Como estrutura de altíssima velocidade, registradores ocupam área valiosa e consomem energia; por isso, mantê-los em baixa quantidade contribui para viabilizar o custo e a frequência do processador, sem inviabilizar a hierarquia restante.

(F) O pipeline de dados pode ignorar completamente os registradores, executando operações diretamente sobre palavras da DRAM com tempo de acesso equivalente, graças a otimizações de barramento.

(V) Quando a hierarquia está bem ajustada, uma fração muito grande das referências do processador é satisfeita pelos níveis superiores, culminando em operandos que já residem nos registradores ou podem chegar a eles com mínima latência a partir da cache.

(F) Em termos de programação, registradores são invisíveis como as caches: não há qualquer exposição na interface de execução, e o compilador não pode considerar seu uso ao gerar código.

(V) A proximidade física e lógica dos registradores em relação às unidades funcionais justifica seu papel como "nível zero" da hierarquia, reduzindo trajetos e atrasos para valores que mudam a cada ciclo ou poucos ciclos.

(F) Para garantir consistência com múltiplas caches, é suficiente atualizar registradores remotos quando um núcleo escreve em seus próprios registradores, pois eles refletem o estado global de memória.

(V) A quantidade limitada de registradores força uma seleção criteriosa do que permanece "à mão" da CPU, enquanto o restante do contexto ativo do programa circula por cache e DRAM, seguindo padrões de localidade e substituição em níveis inferiores.

(F) O espaço de endereçamento de registradores é contínuo e compatível com o endereçamento de bytes e palavras da DRAM, permitindo instruções de leitura/escrita idênticas para ambos os casos.

(V) Do ponto de vista de tecnologia, registradores integram o conjunto de memórias semicondutoras voláteis usadas internamente, compartilhando esse caráter com caches e a memória principal, embora se diferenciem fortemente em latência e custo por bit.

(F) Aumentar arbitrariamente a contagem de registradores é sempre vantajoso, pois melhora desempenho sem qualquer impacto na área, na energia ou na latência do acesso a esse nível.

(V) A hierarquia é desenhada para que palavras "normalmente acessadas" permaneçam nos níveis mais rápidos; os registradores representam o estágio final desse objetivo, onde a latência é mínima e os valores são imediatamente consumidos pelas unidades de execução.

(F) Por serem os mais rápidos, registradores armazenam dados permanentes do usuário, como arquivos e registros de banco de dados, que ficam disponíveis mesmo após desligamentos.

(V) Quando um conjunto pequeno de dados domina a execução por breves períodos, mantê-lo nos registradores evita que o processador “pare” aguardando a DRAM; quando esse conjunto muda, novos valores sobem gradualmente pelos níveis até alcançarem a CPU.

(F) Registradores podem ser ampliados ao tamanho da DRAM sem alterar o tempo de acesso, porque a relação entre capacidade e latência foi superada nas memórias internas modernas.

(V) Como primeiro nível de armazenamento efetivo durante a execução, registradores ajudam a “esconder” a diferença de velocidade entre CPU e memória, compondo, junto com a cache, a estratégia de reduzir o tempo médio de acesso.

(F) A hierarquia poderia prescindir de registradores se a cache fosse suficientemente grande, pois a proximidade física não influencia a latência percebida pelo núcleo.

(V) Apesar de sua função central, registradores não substituem os demais níveis: eles atendem um escopo minúsculo de dados a cada instante, e dependem da cache e da DRAM para abastecimento contínuo conforme os conjuntos ativos mudam durante a execução.

(F) Em multiprocessadores, a coerência global depende de sincronizar conteúdos de registradores entre núcleos, já que eles replicam as mesmas palavras compartilhadas de memória.

(V) A distância entre o ritmo de execução da CPU e os tempos de acesso da DRAM motiva todo o arranjo hierárquico; os registradores situam-se onde essa distância precisa ser praticamente anulada para evitar bolhas no fluxo de instruções.

(F) A semântica de arquivos e registros da memória externa se estende aos registradores para facilitar programação, de modo que operações de E/S podem escrever diretamente neles de maneira persistente.

(V) Em resumo, registradores concentram a computação “imediata” do processador, explorando localidade para manter valores críticos por curtíssimos períodos, enquanto as demais camadas amortizam custos de capacidade e acesso para o restante do estado do programa.

(F) O texto equipara explicitamente a numeração de posições de registrador à de memória principal, afirmando que cada registrador tem “endereço exclusivo” no mesmo espaço linear.

(V) A relação custo/bit crescente e a latência decrescente explicam por que o projeto reserva aos registradores um papel de “ponta da lança” na entrega de dados à execução: pouca capacidade, altíssima velocidade, e cooperação estreita com a cache para manter o núcleo alimentado.

(F) A política de substituição de dados entre registradores e cache depende de algoritmos como LRU ou FIFO implementados diretamente dentro do banco de registradores, já que eles funcionam como uma cache de DRAM.

(V) Quando um novo conjunto de dados passa a dominar a execução, valores anteriores nos registradores perdem relevância e são substituídos por novos operandos trazidos da cache/DRAM, refletindo as mudanças naturais da fase do programa.