8.9 EXERCÍCIOS PROPOSTOS

1. O *pipeline* é uma técnica de projeto de processadores que potencializa enormemente o seu desempenho pela divisão da execução da instrução em diversas etapas. Com relação a isso, responda:
   a) Quais condições devem ser satisfeitas para que o modelo básico de funcionamento do pipeline seja real?
      Resposta: Para que esse modelo básico de funcionamento do pipeline seja real, três condições devem ser satisfeitas. Em primeiro lugar, todos os estágios do pipeline devem ser capazes de executar a sua tarefa específica dentro de um ciclo de relógio. Em segundo lugar, o fluxo de entrada de instruções no pipeline deve ser mantido continuamente, não podendo sofrer interrupções ou atrasos. Finalmente, em terceiro lugar, não devem existir nem dependências de dados (uma instrução precisar do resultado da outra), nem conflitos estruturais (uso de um recurso por mais de um estágio) entre as instruções que estão sendo executadas simultaneamente no pipeline.
   b) Por que o uso de memórias cache é necessários nos processadores com pipeline? Por que normalmente ela é separada em cache de instruções e de dados?
      Resposta: O uso de memórias cache é essencial para atender à alta velocidade de processamento exigida pelos processadores com pipeline, que pode ser k vezes maior do que um processador sem pipeline, sendo k o número de estágios do pipeline. Geralmente, as memórias cache são divididas em cache de instruções e cache de dados para possibilitar ao processador o acesso simultâneo a instruções e dados durante a execução, evitando conflitos no acesso à memória principal entre o estágio de busca de instruções e o estágio de acesso aos dados na memória. Essa divisão permite otimizar o desempenho do processador, garantindo que as instruções e os dados necessários estejam prontamente disponíveis na cache, reduzindo assim o tempo de acesso à memória principal e minimizando gargalos no processamento de dados.
   c) Descreva três modificações arquiteturais necessárias para o uso eficiente do pipeline em um processador.
      Resposta: Algumas modificações são relacionadas a seguir:
      - O uso de memória cache é essencial para atender à maior demanda por instruções imposta pelo pipeline. A memória cache é dividida em instruções e dados, permitindo que o pipeline realize a busca simultânea de instruções e a leitura/escrita de operandos, sem conflitos entre os estágios de busca (B) e acesso à memória (M) do pipeline.
      - O banco de registradores do processador é construído com múltiplas portas para possibilitar a leitura e escrita de vários operandos simultaneamente pelas várias instruções em execução no pipeline.
      - Caminhos de dados alternativos são criados entre os estágios do pipeline para adiantar os resultados produzidos por instruções nos estágios finais para aquelas instruções que estão aguardando pelos operandos nos estágios iniciais. Isso é conhecido como dependência direta de dados entre as instruções.

- Em casos em que o resultado de um desvio ainda não é conhecido, são utilizadas técnicas de predição estática ou dinâmica de desvio para otimizar a busca de instruções. A predição dinâmica baseia-se no comportamento passado do desvio para prever seu comportamento futuro.
- Quando a predição não é possível, pode-se empregar o ”desvio atrasado”, onde o compilador reorganiza as instruções para permitir a execução das instruções após os desvios, otimizando o desempenho da execução do programa.

d) Consideremos um programa com 10.000.000 de instruções. Vamos supor que em uma arquitetura sem pipeline o tempo médio de execução de cada instrução seja de 4 ns. Qual seria o ganho na execução deste programa em um processador com pipeline de 5 estágios com ciclo de relógio de 1 ns?

Resposta: Considerando um programa com 10.000.000 de instruções e um tempo médio de execução de cada instrução de 4 ns em uma arquitetura sem pipeline, o tempo total de execução seria:

Tempo total = (Número de instruções) x (Tempo médio de execução por instrução)

Tempo total = 10.000.000 x 4 ns = 40.000.000 ns = 40 ms

Agora, com um processador com pipeline de 5 estágios e um ciclo de relógio de 1 ns, a taxa de execução é de uma instrução concluída a cada ciclo de relógio, após os 5 ciclos iniciais gastos para a primeira instrução atravessar o pipeline:

Instruções concluídas por ciclo = 1

O tempo total de execução do programa usando o pipeline pode ser calculado apenas pelo número de instruções e o ciclo de relógio, pois o pipeline ideal não tem penalidades:

Tempo total com pipeline = (Número de instruções) x (Ciclo de relógio) + 5 ns

Tempo total com pipeline = $10.000.000 \times 1$ ns + 5 ns = 10.000.005 ns $\approx$ 10 ms

Portanto, o ganho na execução deste programa usando um processador com pipeline de 5 estágios é de 40 ms (tempo sem pipeline) / 10 ms (tempo com pipeline) = 4 vezes. Este é um cenário ideal, no entanto, é importante destacar que na prática, os pipelines reais podem enfrentar penalidades e desafios que podem diminuir o seu desempenho.

2. Com relação ao pipeline, assinale as afirmativas abaixo como verdadeiras ou falsas.
   a) A duração do ciclo de relógio de um processador deve ser menor ou igual ao tempo que o estágio mais lento do pipeline leva para realizar suas operações;
   b) Deve-se procurar dividir a execução da instrução em estágios que tenham o mesmo tempo de execução, para otimizar o desempenho do pipeline;
   c) O pipeline deve ser mantido sempre “cheio”, ou seja, com instruções úteis em todos os seus estágios, para que o seu desempenho máximo seja alcançado.
   d) Cada instrução, individualmente, gasta um tempo igual ou menor para ser executada em um processador com pipeline quando comparado à execução sem pipeline.

   Resposta:
   a) Falso, deve ser maior ou igual.
   b) Verdadeiro.
   c) Verdadeiro.
   d) Falso. Gasta um tempo maior ou igual.

3. O *superpipeline* aprofunda a técnica de *pipeline* com a divisão dos estágios de um processador em estágios ainda menores. Com relação a isso responda:
   a) Descreva brevemente a técnica de *superpipelining*. Como se compara o seu custo de implementação em relação às arquiteturas com pipeline convencional?
   Resposta: A técnica de superpipelining é uma evolução da técnica de pipeline convencional que divide os estágios do processador em estágios ainda menores, permitindo que as instruções sejam divididas em partes menores e se movam mais rapidamente pelo pipeline. Isso aumenta a quantidade de instruções que podem ser processadas simultaneamente e, portanto, aumenta o desempenho geral do processador. Em relação às arquiteturas baseadas em estruturas de pipeline convencionais, o custo de hardware das arquiteturas superpipeline é acrescido por duas razões. A primeira delas é a necessidade de uso de um número maior de registradores para isolar cada sub-estágio do superpipeline dos seus vizinhos. A segunda é a necessidade de uso de uma tecnologia que suporte relógios de frequência mais alta, com bom controle de diferenças de atraso (*skew*) entre os instantes em que uma mesma transição do relógio é percebida em partes distantes no layout físico do processador. Esse problema é diretamente proporcional ao número de estágios, já que isso implica no aumento da frequência do relógio.
   b) Em que situações o aumento da profundidade do *pipeline* pode prejudicar o desempenho do processador?
   Resposta: O aumento da profundidade do pipeline na técnica de superpipelining pode resultar em um desempenho prejudicado devido a situações que levam ao ”esvaziamento” do pipeline. Essas situações incluem falhas no acesso à cache de instruções ou predições incorretas de desvios condicionais. Quando ocorre uma predição incorreta de desvios condicionais, as instruções já presentes no pipeline precisam ser descartadas, e o pipeline deve ser preenchido novamente com as instruções corretas a serem executadas. Esse processo de reabastecimento do pipeline com as instruções corretas resulta em perda de tempo, e o tempo perdido aumenta na mesma proporção que o aumento da profundidade do pipeline.
   c) Qual o impacto do uso da técnica de *superpipelining* na potência dissipada pelos processadores?
   Resposta: O uso da técnica de superpipelining pode aumentar a potência dissipada pelos processadores, especialmente devido à maior frequência de operação necessária para alcançar ciclos de relógio menores e aumentar o desempenho. O aumento da frequência pode resultar em maior consumo de energia e geração de calor, o que requer soluções de resfriamento mais robustas. Além disso, a ocorrência de bolhas e a necessidade de tratar dependências complexas podem resultar em desperdício de energia devido a ciclos ociosos e execução especulativa de instruções desnecessárias. Portanto, o superpipelining exige uma abordagem cuidadosa para garantir o equilíbrio entre desempenho e eficiência energética.
4. Os processadores superescalares oferecem um desempenho significativamente superior às arquiteturas convencionais pela execução de mais de uma instrução por ciclo de relógio. Com relação a isso, responda:
   a) Quais as condições que as instruções devem atender para serem enviadas para execução

nas unidades funcionais em uma arquitetura superescalar?

Resposta: As instruções somente são enviadas para execução nas unidades funcionais quando atendem pelo menos duas condições: primeiro, não violam as regras de dependência de dados, ou seja, todos os seus operandos devem estar prontos e disponíveis; segundo, não há conflito estrutural, devendo existir pelo menos uma unidade funcional disponível que possa executar a instrução sendo despachada. De uma forma geral, as instruções não devem possuir dependências de dados entre si, permitindo que sejam executadas de forma independente e em paralelo; não devem possuir dependências de controle, como desvios condicionais que não foram ainda resolvidos e as unidades funcionais necessárias para a execução das instruções devem estar disponíveis para evitar conflitos de recursos.

b) Como é feito o escalonamento das instruções em uma arquitetura superescalar?

Resposta: O escalonamento de instruções em uma arquitetura superescalar é realizado pelo hardware do processador, que analisa as dependências de dados e controle entre as instruções disponíveis no pipeline. O objetivo é encontrar instruções independentes que possam ser executadas em paralelo nas unidades funcionais disponíveis. As instruções escalonadas são então enviadas para execução nas unidades adequadas.

c) Qual a função da Trace Cache nos processadores superescalares?

Resposta: A Trace Cache é uma estrutura de cache especializada em armazenar sequências de instruções já decodificadas e escalonadas. Ela constrói sequências de instruções, ordenadas pela execução do programa, chamadas de traces. Isso permite que o destino de um desvio seja incluído na mesma linha da Trace Cache onde está o próprio desvio, mesmo que o desvio e suas instruções de destino estejam em endereços distantes um do outro na memória principal. Ela ajuda a aumentar o desempenho dos processadores superescalares ao fornecer instruções escalonadas prontas para execução, reduzindo a carga sobre o decodificador de instruções e acelerando o processo de escalonamento.

d) Como é realizada e qual a importância da predição dinâmica de desvios nos processadores superescalares?

Resposta: A predição dinâmica de desvios é uma técnica utilizada nos processadores superescalares para tentar antecipar o resultado de desvios condicionais. Isso permite que o processador especulativamente siga o caminho correto do fluxo de controle, evitando atrasos causados pela resolução tardia de desvios. A importância dessa técnica é reduzir a penalidade de desvios e manter o pipeline de instruções ocupado com instruções especulativas, que são posteriormente confirmadas ou descartadas quando a resolução dos desvios é conhecida.

e) Qual a finalidade e quais as formas de implementação da janela de instruções nos processadores superescalares?

Resposta: A janela de instruções é uma estrutura de armazenamento temporário no processador que isola os estágios de busca e decodificação dos estágios de execução propriamente ditos, e onde as instruções possam aguardar pelos seus operandos e por unidades funcionais disponíveis, permitindo que elas sejam emitidas e executadas fora de ordem, conforme disponibilidade de recursos. A finalidade da janela de instruções é aumentar a capacidade de processamento e manter o pipeline ocupado, evitando atrasos causados por

dependências de dados ou desvios. Essa janela pode ser implementada de forma centralizada ou distribuída, ou seja, pode ser uma única fila comum a todas unidades funcionais ou existirem diversas filas, uma para cada unidade funcional do processador. No caso da janela centralizada, uma lógica para despacho concorrente das instruções armazenadas na janela de instruções deve ser utilizada para garantir que mais de uma instrução possa ser enviada para as unidades funcionais a cada ciclo.

f) Como é feita a recuperação do estado arquitetural do processador no caso da ocorrência de exceções ou de previsões incorretas de desvios?
Resposta: Quando ocorre uma exceção ou previsão incorreta de desvio em um processador superescalar, é necessário descartar as instruções especulativas e recuperar o estado arquitetural para o ponto correto no fluxo de controle. Esse mecanismo de recuperação é chamado de *buffer* de reordenação (*reorder buffer*), que é basicamente uma fila onde as instruções aguardam a sua retirada na ordem especificada no código original do programa. Assim, as exceções e a verificação do resultado dos desvios preditos só é feita quando as instruções chegam na primeira posição da fila, ou seja, quando todas as instruções anteriores a ela na ordem do programa terminaram a sua execução sem problemas.

g) O que são e como pode ser feita a remoção das dependências falsas de dados nos processadores superescalares?
Resposta: A remoção das dependências falsas de dados é uma técnica importante nos processadores superescalares para evitar conflitos e atrasos causados por dependências que não afetam o resultado final das instruções. A remoção dessas dependências pode ser facilmente realizada através da técnica de renomeação de registradores, que consiste basicamente em alocar um registrador não utilizado para substituir o registrador arquitetural da segunda instrução. Essa técnica pode ser realizada por *software* ou por mecanismos de *hardware*, como a tabela de renomeação.

h) Quais as modificações que devem ser feitas no banco de registradores dos processadores superescalares?
Resposta: O banco de registradores deve ter múltiplas portas de leitura e de escrita, para fornecer os operandos necessários e receber os resultados das diversas instruções em execução. Isso tem um impacto negativo no seu tempo de acesso, que aumenta conforme aumenta o número de portas existentes. Assim, algum tipo de limitação nesse número de portas pode existir, resultando em uma forma de arbitragem para determinar quais instruções terão acesso primeiro aos seus operandos. Isso permite que várias instruções especulativas possam escrever em registradores diferentes sem causar conflitos e sem afetar o estado arquitetural final.

i) Qual a finalidade da fila de acesso à memória nos processadores superescalares?
Resposta: A fila de acesso à memória deve ter suporte para tratamento de dependências de dados entre as instruções de leitura (*load*) e escrita (*store*) em memória, de modo a otimizar os acessos à memória. Assim, as operações de leitura podem ser adiantadas em relação às de escrita, caso não sejam para o mesmo endereço. E, caso haja alguma coincidência de endereços, os dados correspondentes podem ser adiantados da instrução de *store* para a instrução de *load* dependente.

5. As arquiteturas *multicore* têm se estabelecido com um padrão nos modernos processador. Com relação a isso, responda:
   a) Qual a principal motivação para o uso de arquiteturas *multicore*?
      Resposta: A principal motivação para o uso de arquiteturas *multicore* é o aumento do desempenho e eficiência dos processadores. Com o aumento da frequência dos processadores chegando a limites físicos e a complexidade de projetar processadores superescalares cada vez maiores, tornou-se mais difícil e ineficiente aumentar o desempenho simplesmente aumentando a frequência de relógio de um único núcleo. Isso levaria a um aumento do aquecimento dos processadores, exigindo melhorias nos sistemas de refrigeração que não acompanharam o crescente desenvolvimento dos processadores, resultando em um impasse no projeto desses processadores.
      A utilização de múltiplos núcleos em um único circuito integrado permite que várias tarefas sejam executadas em paralelo, aumentando significativamente a capacidade de processamento total. Além disso, os processadores multicore não demandam um esforço significativo de engenharia a cada geração, pois cada família de processadores requer apenas replicar cópias adicionais do núcleo do processador, com algumas modificações nas conexões lógicas. Isso evita a necessidade de redesenhar todo o projeto do núcleo dos processadores.
      Dessa forma, os modelos de *multicore* mantêm sua essência à medida que evoluem, aumentando principalmente o número de núcleos presentes nos integrados. Essa abordagem permite melhorar o desempenho do processador sem depender apenas do aumento da frequência de relógio, resultando em processadores mais eficientes e poderosos.
   b) Quais as vantagens na utilização de arquitetura *multicore*?
      Resposta: A execução simultânea de várias tarefas em núcleos separados aumenta o desempenho total do processador e permite que mais tarefas sejam concluídas em menos tempo. Em vez de aumentar a frequência de relógio para melhorar o desempenho, os processadores multicore podem alcançar maior eficiência energética ao executar tarefas em paralelo em núcleos com frequências menores. Em um processador multicore, se eventualmente um núcleo falhar, os outros núcleos podem continuar funcionando, tornando a arquitetura mais resiliente.
   c) Quais as vantagens do uso de arquiteturas *multicore* na programação *multithreading*?
      Resposta: Com o uso de arquiteturas multicore, a programação multithreading se torna mais eficiente e acessível. Os desenvolvedores podem criar aplicativos que aproveitam o paralelismo inerente da arquitetura multicore, dividindo tarefas em várias threads que podem ser executadas em núcleos diferentes. Isso permite melhor utilização do hardware e, portanto, maior desempenho em sistemas que possuem processadores multicore. Hoje em dia, em qualquer multicore com memória cache compartilhada de nível 3 (L3), cada evento de comunicação toma poucos ciclos do processador. Com a latência dessa forma, os atrasos de comunicação têm muito menos impacto no desempenho do sistema, sendo que os programadores ainda devem dividir seus trabalhos em threads paralelas, porém não precisam se preocupar tanto com a independência dessas threads, já que o custo de comunicação é relativamente baixo.

d) Qual a vantagem do uso de *multicore* no projeto de novas gerações de processadores?
Resposta: O uso de multicore no projeto de novas gerações de processadores permite que os fabricantes continuem aumentando o desempenho geral dos processadores sem depender exclusivamente do aumento da frequência de relógio. Apenas com o aumento do número de núcleos, sem um grande esforço de engenharia, é possível aumentar o poder de processamento geral e atender às demandas de aplicações cada vez mais complexas, como inteligência artificial, aprendizado de máquina, computação gráfica avançada e processamento de grandes volumes de dados. Os modelos de processadores multicore mantêm a sua essência de acordo com sua evolução, aumentando praticamente somente o número de núcleos presente nos chips. O projeto das placas do sistema necessita de uma menor mudança em relação às gerações dos multicores, sendo que a única real diferença é que as placas necessitam tratar da maior largura de banda de E/S que é exigida por esses sistemas.

e) Quais vantagens do ajuste do ponto de trabalho de cada núcleo do processador?
Resposta: O ajuste do ponto de trabalho de cada núcleo do processador permite que o sistema otimize o consumo de energia e a eficiência de acordo com a carga de trabalho em tempo real. Isso significa que cada núcleo pode ajustar dinamicamente sua frequência de relógio e tensão para corresponder à demanda de processamento no momento. Quando a carga de trabalho é baixa, alguns núcleos podem reduzir a frequência para economizar energia, enquanto outros podem aumentar a frequência para lidar com tarefas mais exigentes. Esse ajuste dinâmico resulta em uma melhor eficiência energética do sistema como um todo.

6. As arquiteturas VLIW são um paradigma alternativo para a execução de múltiplas instruções por ciclo de relógio. Com relação a isso, responda:

a) Qual a diferença na estratégia utilizada nas arquiteturas VLIW para a execução simultânea de várias instruções?
Resposta: Nas arquiteturas VLIW (*Very Long Instruction Word*), a estratégia para a execução simultânea de várias instruções é realizar o escalonamento de instruções em tempo de compilação. Isso significa que o compilador agrupa múltiplas instruções em uma única instrução longa chamada ”palavra VLIW”, que contém várias operações independentes. Cada palavra VLIW é então executada em paralelo pelo *hardware* do processador, aproveitando as unidades funcionais disponíveis.

b) Qual a principal desvantagem das arquiteturas VLIW?
Resposta: Uma desvantagem das arquiteturas VLIW é a dificuldade de encontrar um número ideal de instruções para cada palavra VLIW que seja eficiente em todas as situações. Isso acontece porque o escalonamento de instruções é feito em tempo de compilação e não pode levar em consideração as condições reais de execução, como dependências de dados e desvios condicionais. Se a palavra VLIW contiver instruções que dependem de resultados ainda não disponíveis ou incluir instruções desnecessárias, a eficiência da execução pode ser prejudicada, resultando em um processador subutilizado. Outra desvantagem das arquiteturas VLIW está relacionada com a falta de compatibilidade binária com o código das arquiteturas convencionais. Mais grave ainda, basicamente cada nova implementação de

uma arquitetura VLIW requer a total recompilação do código das aplicações.

c) Quais as vantagens do escalonamento estático de instruções realizado em tempo de compilação pelas arquiteturas VLIW?

Resposta: O escalonamento estático de instruções realizado em tempo de compilação pelas arquiteturas VLIW apresenta como vantagem principal a simplicidade do *hardware*: Como o escalonamento é feito pelo compilador, o *hardware* do processador pode ser mais simples, sem a necessidade de mecanismos complexos de detecção e resolução de dependências de dados e desvios condicionais em tempo de execução, pois tudo foi resolvido em tempo de compilação.

d) Historicamente, quando e quais foram as primeiras iniciativas do desenvolvimento de arquiteturas VLIW?

Resposta: Duas empresas foram fundadas em 1984 para construir computadores com tecnologia VLIW: Multiflow e Cydrome. A Multiflow foi iniciada por Fisher e seus colegas da Universidade de Yale, sendo que a Cydrome foi fundada por Bob Rau, que foi um outro pioneiro da VLIW e seus colegas. Em 1987 a Cydrome lançou o seu primeiro processador comercial, o Cydra 5, com uma palavra de 256 bits e incluía suporte em hardware para a técnica de software pipeline. No mesmo ano a Multiflow lançou a Trace/200, com uma palavra de 256 bits, para um despacho de até 7 operações por ciclo. Infelizmente as primeiras máquinas VLIW foram um fracasso comercial, o que levou ao fechamento da Cydrome em 1998 e da Multiflow em 1990. Essas primeiras iniciativas ajudaram a estabelecer as bases conceituais para arquiteturas VLIW e contribuíram para o desenvolvimento de técnicas de escalonamento estático de instruções.

7. As arquiteturas *multithreading* apresentam um *hardware* especializado para a execução simultânea de diversas *threads*. Com relação a isso, responda:

a) Qual o contexto característico de uma *thread*?

Resposta: O contexto de uma *thread* inclui o estado atual de execução da *thread*, como os valores dos registradores, apontador de instruções, a pilha e outros elementos necessários para retomar a execução dessa thread a partir de onde ela foi interrompida. Em outras palavras, o contexto de uma *thread* representa sua posição e progresso dentro do programa em execução.

b) Qual a origem das operações de longa latência em um processador?

Resposta: As operações de longa latência em um processador podem ser causadas por diversos fatores, tais como o acesso à memória principal, acesso a dados e instruções em uma memória remota, e a espera por operações de sincronização no acesso a dados compartilhados, operações de ponto flutuante complexas ou dependências entre instruções, entre outros cenários. Essas operações podem demandar um tempo significativo para serem concluídas, resultando em atrasos na execução do programa.

c) Como as arquiteturas *multithreading* reduzem o efeito negativo da operações de longa latência realizadas pelo processador?

Resposta: As arquiteturas *multithreading* buscam reduzir o efeito negativo das operações de longa latência usando a técnica de troca de contexto. Quando uma *thread* encontra uma operação de longa latência, o processador pode mudar a execução para outra *thread*

que não esteja bloqueada, permitindo que outra tarefa seja executada enquanto a primeira está esperando o resultado. Dessa forma, o processador pode aproveitar melhor seu tempo de execução, aumentando a utilização dos recursos e melhorando o desempenho geral.

d) O que caracteriza o modelo de *multithreading* de granulosidade fina?
Resposta: No modelo de *multithreading* de granulosidade fina, o processador alterna a cada ciclo entre instruções de *threads* diferentes. Isso significa que cada instrução nova é buscada e executada sequencialmente em diferentes *threads*. Com isso a lógica de controle do *pipeline* é bastante simplificada, pois não existem dependências de dados e de controle, e, consequentemente, o *pipeline* pode ser bem mais rápido. Além disso, neste modelo, a sobrecarga para troca de contexto é nula, já que o processador sempre sabe antecipadamente de qual contexto será a próxima instrução a ser executada e nenhuma instrução já presente no *pipeline* precisa ser descartada no momento de uma troca de contexto.

e) Qual o número mínimo de contextos que devem suportados em *hardware* nas arquiteturas com granulosidade fina?
Resposta: Nas arquiteturas de granulosidade fina, o número mínimo de contextos suportados é igual ao número de estágios do pipeline. Cada estágio do pipeline é associado a uma thread, e a troca de contexto ocorre a cada estágio, permitindo a execução simultânea de várias threads.

f) Qual a principal desvantagem da abordagem de granulosidade fina para a implementação de *multithreading*?
Resposta: Para que o esquema de *multithreading* de granulosidade fina possa operar, o número de contextos suportados em *hardware* deve ser, no mínimo, igual ao número de estágios do *pipeline*. Entretanto, como muitas vezes certos contextos podem não ter uma instrução pronta para ser executada, em geral, é necessário um número bem maior de contextos para tornar a arquitetura efetiva, isto é, uma arquitetura em que sejam pouco frequentes os ciclos onde o *pipeline* não possa ser preenchido com instruções de algum contexto. A principal desvantagem desta abordagem para a implementação de *multithreading* é que o desempenho de código sequencial (código com uma única *thread*) pode ser bastante ruim, já que cada instrução desta única *thread* é executada em tantos ciclos quantos forem o número de estágios do pipeline.

g) Como se dividem as arquiteturas *multithreading* de granulosidade grossa? Descreva os mecanismos de troca de contexto em cada caso.
Resposta: As arquiteturas dessa classe podem ser divididas em estáticas e dinâmicas. As arquiteturas estáticas podem contar com um mecanismo implícito ou explícito para a troca de contexto. No caso implícito, a troca de contexto ocorre quando certas instruções (como load, store, branch, etc.) são encontradas. Já no caso explícito, a troca de contexto é realizada quando instruções específicas de troca de contexto são encontradas. O *overhead* da troca de contexto é de apenas um ciclo se a instrução for descartada no estágio de busca e zero se for executada. Por outro lado, nas arquiteturas dinâmicas, o processador realiza a troca de contexto quando ocorre uma operação de longa latência, como uma falha na cache ou operações de sincronização. O custo da troca de contexto é proporcional à profundidade do pipeline no estágio que aciona essa troca de contexto, pois as instruções

posteriores no pipeline devem ser anuladas. Essas abordagens de troca de contexto são utilizadas para otimizar o desempenho dos processadores e garantir uma transição eficiente entre diferentes contextos de execução, minimizando o impacto no processamento e garantindo um melhor aproveitamento dos recursos do processador.

h) Qual a dificuldade para execução de aplicações com alta frequência de operações de grande latência em arquiteturas *multithreading* de granulosidade grossa?
Resposta: Quando essas operações de alta latência ocorrem com frequência elevada, a técnica de *multithreading* enfrenta dificuldades para sobrepor a execução em um contexto com operações de longa latência de vários outros contextos. Nesse cenário, a capacidade de ocultar o tempo gasto nessas operações torna-se limitada, afetando a eficácia do *multithreading* como uma estratégia de otimização de desempenho. É essencial, portanto, analisar cuidadosamente o comportamento da aplicação e a frequência das operações de alta latência para determinar se o *multithreading* de granulosidade grossa é a abordagem mais adequada para melhorar o desempenho do processador.

8. As arquiteturas *simultaneous multithreading* são um tipo de arquitetura que permitem a execução simultânea de várias *threads* em um processador superescalar. Com relação a isso, responda:
   a) Enumere as principais mudanças na arquitetura do processador que são necessárias para o suporte ao *multithreading* simultâneo.
   Resposta: Para suportar o *multithreading* simultâneo, o processador requer algumas mudanças em sua arquitetura, incluindo ter um grande banco de registradores para armazenar o contexto de cada *thread* em execução, permitindo que cada *thread* tenha seu próprio conjunto de registradores, além de registradores adicionais para renomeação. Deve também compartilhar as unidades de execução entre as *threads* para otimizar o uso dos recursos do processador. Além disso, a arquitetura pode incluir múltiplos apontadores de instrução, dois estágios no *pipeline* para acesso aos registradores, várias pilhas para predição do endereço de retorno das rotinas (uma para cada *thread*), tabelas de renomeação individualizadas para cada *thread*, e a identificação de cada *thread* nas TLBs, nos mecanismos de predição de desvio e nas janelas de instrução.
   b) Em que tipos de aplicação o uso de *multithreading* simultâneo é especialmente benéfico?
   Resposta: O *multithreading* simultâneo é especialmente benéfico em aplicações que possuem um alto grau de paralelismo, ou seja, são capazes de executar múltiplas tarefas independentes ao mesmo tempo. Algumas aplicações que podem se beneficiar significativamente do *multithreading* simultâneo incluem ambientes comerciais onde a velocidade de uma transação individual não é tão importante quanto o número total de transações realizadas. Espera-se que o SMT aumente a vazão (*throughput* em inglês) das tarefas com conjuntos de trabalho grandes ou que mudam com frequência, como servidores de banco de dados e servidores da Web, assim como aquelas tarefas caracterizadas por altas taxas de falhas da cache, que tendem a usar os recursos de processador e memória de forma inadequada, com maiores latências para serem ocultadas.
   c) Como o uso do *multithreading* simultâneo afeta as estruturas do processador compartilhadas pelas diversas *threads* em execução?
   Resposta: O uso do *multithreading* simultâneo afeta as estruturas compartilhadas do pro-

cessador, como a memória cache, tabela de renomeação de registradores, janela de instruções, *reorder buffer*, e as tabelas de tradução de endereços. O compartilhamento de recursos entre as *threads* pode resultar em maior concorrência por essas estruturas compartilhadas, levando a possíveis conflitos, contenção e atrasos. Para mitigar esses efeitos, o tamanho dessas estruturas deve ser aumentado, permitindo a identificação das instruções de cada *thread*.

9. As arquiteturas paralelas têm como objetivo alcançar níveis diferenciados de desempenho com o uso de múltiplos processadores para a execução em paralelo das aplicações. Com relação a isso, responda:
   a) O que são aceleradores? Exemplifique e descreva alguns dispositivos deste tipo.
      Resposta: Aceleradores são dispositivos ou unidades especializadas que são projetados para executar tarefas específicas de forma mais eficiente e rápida do que o processador principal de um computador. Eles trabalham em conjunto com o processador para melhorar o desempenho geral do sistema em tarefas específicas. Exemplos de aceleradores incluem:
      - Unidades de Processamento Gráfico (GPUs): As GPUs são projetadas para executar cálculos gráficos intensivos usados em jogos, animações, modelagem 3D e outros aplicativos que requerem processamento paralelo de grande volume de dados. Além disso, as GPUs também são usadas em computação de alto desempenho (HPC) para acelerar cálculos científicos complexos.
      - Processadores Tensor: São aceleradores otimizados para cálculos de aprendizado de máquina e inteligência artificial, especialmente redes neurais profundas (deep learning). Eles lidam eficientemente com operações matriciais, fundamentais para essas aplicações.
      - FPGAs (Field-Programmable Gate Arrays): São dispositivos programáveis que podem ser configurados para executar tarefas específicas em paralelo, tornando-os adequados para aplicações que requerem alta performance e baixa latência, como processamento de sinal, criptografia e outros.
   b) Quais são e quais as formas de programação resultantes dos dois grandes tipos de arquiteturas paralelas?
      Resposta: Existem dois grandes tipos de arquiteturas paralelas e, consequentemente, duas formas de programação associadas a elas:
      - Memória Compartilhada (*Shared Memory*): Nesse tipo de arquitetura, vários processadores compartilham um espaço de memória comum, permitindo que eles acessem e modifiquem os mesmos dados. A programação em memória compartilhada é geralmente mais simples, pois os processadores podem se comunicar por meio de variáveis compartilhadas.
      - Memória Distribuída (*Distributed Memory*): Nesta arquitetura, cada processador tem sua própria memória local e não pode acessar diretamente a memória dos outros processadores. A programação em memória distribuída é mais complexa, pois os processadores precisam se comunicar explicitamente por meio de trocas de mensagens.
   c) Quais as características principais das arquiteturas de memória compartilhada?

Resposta: As principais características das arquiteturas de memória compartilhada são as seguintes:

- Cada processador tem acesso a um espaço de endereçamento global comum a todos.
- Esse tipo de arquitetura apresenta como vantagem não necessitar do particionamento de código ou dados, logo técnicas de programação sequenciais podem ser facilmente adaptadas.
- A sincronização entre os processadores é facilitada, permitindo a comunicação através de variáveis compartilhadas.
- Não há também necessidade da movimentação física dos dados, quando dois ou mais processadores se comunicam, resultando em uma comunicação entre processos ou *threads* bastante eficiente.

As suas desvantagens são a necessidade do uso de primitivas especiais de sincronização quando do acesso a regiões compartilhadas de memória, para assegurar um resultado correto para a computação, além da falta de escalabilidade devido ao problema de contenção de memória. Depois de um determinado número de processadores a adição de mais processadores não aumenta o desempenho.

d) Quais as características principais das arquiteturas de memória distribuída?
Resposta:

- Cada processador tem sua própria memória local e não pode acessar diretamente a memória de outros processadores.
- Os processadores se comunicam explicitamente trocando mensagens entre si.
- Apresenta como vantagens serem altamente escaláveis e permitirem a construção de processadores maciçamente paralelos.
- A programação em memória distribuída é mais complexa e requer técnicas de comunicação eficientes para garantir o desempenho e a escalabilidade do sistema.

10. A classificação de Flynn tenta estabelecer uma taxonomia para a subdivisão dos diversos tipos de arquitetura paralela. Com relação a isso, responda:

a) Descreva e caracterize brevemente as arquiteturas SISD.
Resposta: Arquiteturas SISD (*Single Instruction, Single Data*): Nas arquiteturas SISD, há um único processador que executa uma única instrução em um único dado de cada vez. Isso significa que o processador sequencialmente busca instruções em um fluxo e as executa em um fluxo de dados. É a forma mais básica de processamento e não possui paralelismo real, pois só há um fluxo de instruções sendo executado por um único processador.

b) Descreva e caracterize brevemente as arquiteturas SIMD. Exemplifique os seus principais tipos.
Resposta: Arquiteturas SIMD (*Single Instruction, Multiple Data*): Nas arquiteturas SIMD, uma única instrução é executada simultaneamente em múltiplos dados, utilizando processadores especializados que trabalham em paralelo. Essas arquiteturas são adequadas para aplicações que exigem o processamento de grandes conjuntos de dados idênticos, realizando a mesma operação em todos eles ao mesmo tempo. Entre uma das divisões possíveis dessa classe são os processadores vetoriais, arquiteturas SIMD convencionais, e

arquiteturas sistólicas.

c) Descreva e caracterize brevemente as arquiteturas MIMD. Quais são os seus principais grupos?

Resposta: Arquiteturas MIMD (*Multiple Instruction, Multiple Data*): Nas arquiteturas MIMD, múltiplos processadores independentes executam diferentes instruções em diferentes conjuntos de dados ao mesmo tempo, permitindo o processamento paralelo real. Existem diferentes grupos de arquiteturas MIMD:

- MIMD com Memória Distribuída: Cada processador tem sua própria memória local e se comunica com outros processadores através de trocas de mensagens. Exemplo: Clusters de computadores.
- MIMD com Memória Compartilhada: Múltiplos processadores compartilham uma única memória global, permitindo acesso direto aos mesmos dados. Exemplo: Computadores multiprocessados ou SMP (*Symmetric Multi-Processing*).

d) Descreva e caracterize as arquiteturas MIMD com Memória Distribuída.

Resposta: Arquiteturas MIMD com Memória Distribuída: Nas arquiteturas MIMD com memória distribuída, cada processador tem sua própria memória local e não pode acessar diretamente a memória de outros processadores. A comunicação entre processadores é feita através de troca de mensagens, o que exige uma coordenação mais complexa, mas permite maior escalabilidade e independência entre os processadores.

e) Descreva e caracterize as arquiteturas MIMD com Memória Compartilhada. Quais são os seus principais tipos?

Resposta: Arquiteturas MIMD com Memória Compartilhada: Nas arquiteturas MIMD com memória compartilhada, múltiplos processadores compartilham uma única memória global, permitindo acesso direto aos mesmos dados. A sincronização entre os processadores é mais fácil, pois eles podem se comunicar através de variáveis compartilhadas. Essa abordagem simplifica a programação paralela e é mais adequada para aplicações que requerem comunicação frequente entre os processadores. Exemplos desse tipo de arquitetura são as arquiteturas UMA e NUMA:

- Arquiteturas UMA: São arquiteturas com memória única global. O tempo de acesso à memória é uniforme para todos os nós de processamento e, normalmente, os nós de processamento e a memória são interconectados através de barramento único. Possuem um número reduzido de nós de processamento e a coerência das caches é mantida por hardware com o uso da técnica de *snooping*. Essas arquiteturas são conhecidas também como multiprocessamento simétrico (SMP) ou multiprocessamento em chip (CMP ou multicore).
- Arquiteturas NUMA: Nessas arquiteturas a memória é dividida em tantos blocos quanto forem os processadores do sistema, e cada bloco de memória é conectado via barramento a um processador com memória local. O acesso aos dados na memória local é muito mais rápido do que o acesso aos dados em blocos de memória remotos.