Disciplina SSC0510
Arquitetura de Computadores

Grupo N°: 5

Nome:	**Pedro Afonso Fazio Michalichem**	**N° USP: 10734196**
Nome:	**Caio Ohman Balthazar**	**N° USP: 10415227**
Nome:	**João Paulo Garcia**	**N° USP: 11816056**
Nome:	**Ricardo Atakiama**	**N° USP: 1026282**

1ª Questão: Explique o que são, compare e exemplifique as arquiteturas SISD, SIMD, MISD, MIMD.

A taxonomia de Flynn classifica as arquiteturas de computadores de acordo com o fluxo de instruções e fluxo de dados que um computador executa uma sequência de instruções.

Na classe **Single Instruction Single Data stream (SISD)**, há apenas uma instrução sendo processada em uma única unidade de controle operando sobre um único dado por vez. Desta forma, o programa segue uma organização puramente sequencial e nenhum paralelismo é explorado. Como exemplo tem-se os computadores pessoais mais convencionais e mainframes.

Já na **Single Instruction Multiple Data stream (SIMD)**, uma única instrução é aplicada simultaneamente a diferentes porções de dados. Nesse contexto, o programa ainda segue uma organização sequencial. Essa classe serve bem a computação científica, que envolve operações com vetores ou matrizes. O Pentium III marca a inclusão de instruções para SIMD, as SSE (Streaming SIMD Extensions). Uma aplicação que ilustra este conceito é o processamento de uma imagem RGB para escala cinza. Cada pixel da imagem, que contém as componentes vermelho, verde e azul passará por uma mesma operação de transformação para resultar na nova imagem em escala cinza.

A classe **Multiple Instruction Single Data stream (MISD)**, por sua vez, é teórica e representa máquinas com múltiplas unidades de controle que operam instruções distintas sobre um mesmo dado. Máquinas desse tipo não apresentaram muitas aplicações e não foram exploradas comercialmente.

Por fim, **Multiple Instruction Multiple Data stream (MIMD)**, representa máquinas com múltiplas unidades de controle, operando instruções diferentes em dados diferentes. Em oposição a SIMD, as máquinas MIMD são assíncronas, porque cada fluxo de instrução é independente da outra. Por exemplo, um processador pode estar trabalhando com aplicações diferentes, como um núcleo processa a imagem, outro o áudio, outro o sistema operacional.

2ª Questão: Explique o que são, compare e exemplifique arquiteturas com memória compartilhada e memória distribuída.

Arquiteturas com memórias compartilhadas tem, por definição, um barramento de memória utilizando múltiplos processadores. Tipicamente, os processadores não se comunicam diretamente sendo a memória a ponte entre eles.

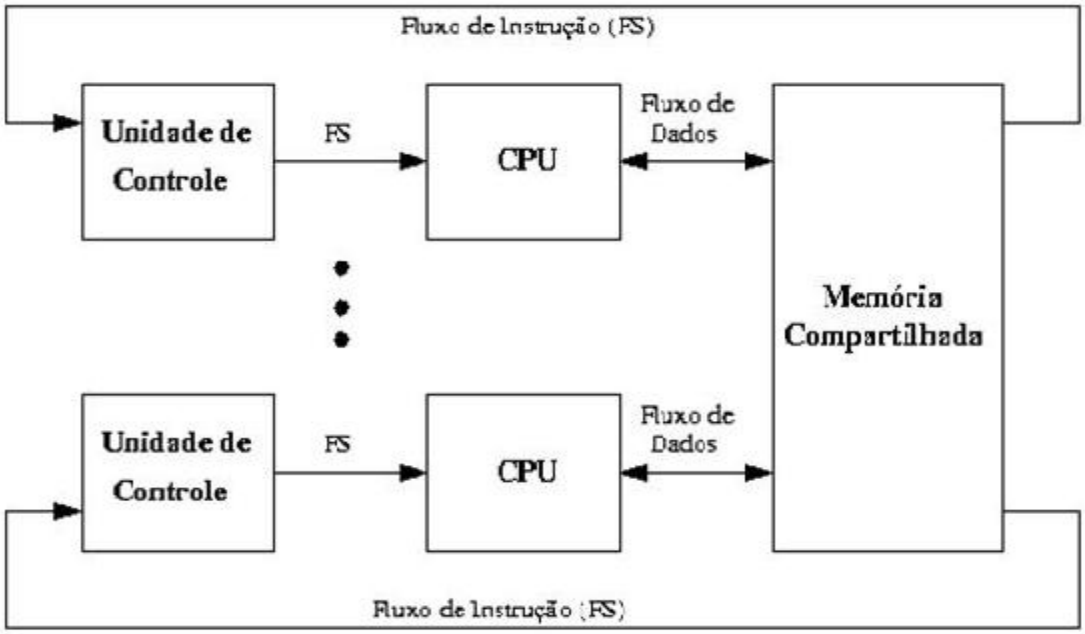

Figura 1: diagrama de memória compartilhada

Arquiteturas com memória distribuída se baseiam no processador possuindo a sua própria memória e há uma rede de interconexão juntamente a uma interface de entrada e saída quando há mais processadores.

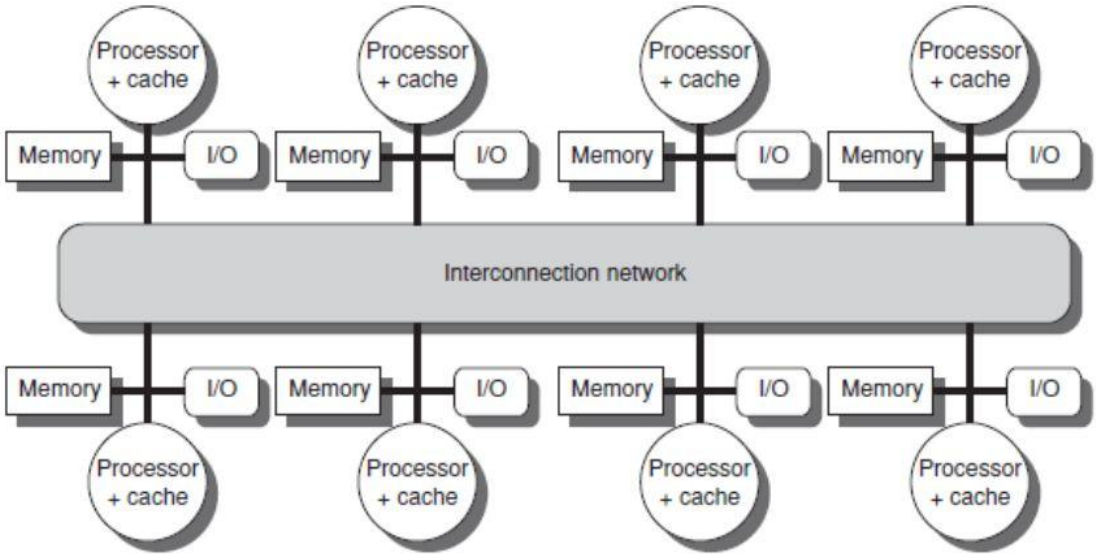

Figura 2: Memória distribuída

A maior diferença é que ao executar uma operação de Load/Store nas arquiteturas de memórias compartilhadas há o acesso direto a toda a memória. No caso das arquiteturas de memória distribuída, se desejar acessar um dado que não está no processador da respectiva memória, tem que realizar uma operação de entrada e saída que consiste em escrever o dado na rede de interconexão, tem que ser lido pelo outro processador que localiza o dado do próprio banco e manda de volta para a rede de interconexão e, finalmente, é lida pelo processador. Uma desvantagem das arquiteturas de memória compartilhada é que não há um sistema implícito de sincronização dos processos sendo necessário, por exemplo, o uso de semáforos para esse controle de corridas.

Um exemplo de arquiteturas de memória distribuída é o cluster. O cluster é um conjunto de nós controlado por uma central. Uma grande vantagem do cluster é sua relação preço com performance. O exemplo da memória compartilhada é dentro dos PC(Computadores Pessoais) com diversos núcleos acessando a mesma memória.

3ª Questão: Explique o que são, compare e exemplifique as seguintes máquinas:

1. Processador com pipeline de operações

Neste tipo, os estágios de busca de instrução em memória, decodificação das instruções dos registradores, execução da instrução, acesso a um operando da memória e escrita do resultado no registrador, podem estar ocorrendo simultaneamente, aumentando a produtividade do sistema. Isso é possível porque as tarefas utilizam recursos diferentes. Neste caso, pode-se obter maior ganho de

performance havendo um número maior de estágios, pois quanto maior o número de estágios maior o número de instruções são processadas em um determinado instante. No entanto, quanto maior o número de estágios pode haver também uma maior dependência entre esses estágios, como a decodificação de uma instrução em um registrador pode ocorrer junto com a escrita de um resultado no mesmo registrador e por isso deve haver um tratamento desses conflitos. Diferentemente dos processadores superescalares, não há mais de uma busca ocorrendo em um mesmo ciclo.

2. Processadores Superescalares

Nos processadores superescalares, há replicação dos componentes internos do processador explorando o paralelismo a nível de instrução de modo que se possa colocar várias instruções em cada etapa do pipeline. Essas instruções são iniciadas simultaneamente e executadas independentemente umas das outras.

Componentes especiais garantem a execução de mais de uma instrução por ciclo. São eles a **Unidade de Busca de Instruções**, que contém um algoritmo de predição de desvio, e é muito importante que ele tenha alta taxa de acerto para continuar executando sua função sem ter que esperar os resultados dos desvios. Além disso, há uma **Unidade de Decodificação**, que faz a leitura de vários operandos do banco de registradores por ciclo. E também apresenta **Unidades Funcionais Inteiras e de Ponto Flutuante**, que precisam haver em quantidade suficiente para executar as 'n' instruções buscadas e decodificadas a cada ciclo, conforme foram projetadas. Alguns exemplos de processadores superescalares são Intel Pentium Pro e Pentium II, AMD K5, K6, e Athlon.

3. Processadores Paralelos

São multiprocessadores que exploram o paralelismo a nível de processo, em que os processos são alocados por uma política de escalonamento para serem executados cada um em um core. Outra forma seria um programa rodando em vários processadores. A coordenação entre os processadores paralelos ocorre por meio da sincronização usando locks. Quando uma thread precisa acessar uma região de memória compartilhada, ele precisa ganhar acesso ao lock para poder fazer a escrita. Depois de usar, a thread libera esse lock para outra thread poder usar. O Xeon e5345 quad-core é um exemplo de processador paralelo, assim como os Intel Core i3, i5, i7 e i9.

4ª Questão: Explique as limitações intrínsecas do paralelismo: Dependência de dados, Dependência de desvio, Conflito de recurso (ULA) e o que pode ser feito para minimizar esses problemas.

As práticas do paralelismo implicam em certas limitações intrínsecas, sendo elas: **Dependência de dados, Dependência de desvio, Conflito de recurso (ULA)**. Para exemplificar melhor sobre cada uma, elas serão separadas por tópicos

Dependência de dados: Podemos usar um exemplo para deixar claro essa limitação da seguinte forma: segunda instrução pode ser buscada e decodificada antecipadamente, mas não pode ser executada pois depende que a primeira instrução tenha sua execução completada. Também pode ser necessário que uma etapa da instrução seguinte seja atrasada até que seja resolvida a dependência

Dependência de desvio: A presença de desvios condicionais acaba prejudicando a operação da pipeline. Sendo que a instrução seguinte a um desvio condicional, tomado ou não, depende dessa instrução de desvio, e não pode ser executada até que seja completada a execução da instrução de desvio Em um pipeline superescalar, por exemplo, o número de instruções perdidas a cada atraso é maior

Conflito de recurso (ULA): Ocorre quando duas ou mais instruções tentam acessar ao mesmo tempo um mesmo recurso (memória, barramentos, unidades funcionais, etc.)

Para minimizar alguns desses problemas, podemos auxiliar no conflito de recursos, que diferente das dependências de dados que não podem ser eliminadas, esses conflitos de recursos podem ser superados pela duplicação dos mesmos, porém isso causará impactos no custo e complexidade do projeto. Também para a dependência procedural, se considerar instruções de tamanhos fixos não haverá problemas maiores pois a dependência procedural pode ser estimada, entretanto quando se trata de instruções com tamanhos variados acaba-se abrindo a possibilidade de gerar um novo tipo de dependência. E para a dependência de dados, uma maneira de compensar este atraso é o compilador reordenar as instruções para que uma ou mais instruções subsequentes que não dependem da leitura de memória possam seguir pelo pipeline. Este esquema é menos eficiente no caso de um pipeline superescalar: as instruções independentes executadas durante a leitura provavelmente serão executadas no primeiro ciclo da leitura, deixando o processador sem fazer nada até que a leitura se complete.

5ª Questão: Explique renomeação de registradores.

Na arquitetura de computadores, a Renomeação de Registradores refere-se a uma técnica utilizada para evitar a desnecessária serialização das operações de um programa, imposta pelo uso dos registradores por essas operações. Levando em

consideração que as instruções não podem simplesmente serem executadas serialmente, pois isso seria custoso em termos de tempo, utiliza-se a técnica da Renomeação de Registradores. Esta técnica consiste em renomear os registradores, aumentando o paralelismo disponível, ou seja, permitindo que duas instruções que antes deveriam ser executadas de maneira serial possam agora ser executadas concorrentemente pois estão utilizando registradores diferentes. Então basicamente eles são um método para lidar com conflitos de armazenamento causados pelas dependências de saída e anti dependências. Registradores são alocados dinamicamente pelo processador, sendo associados aos valores requeridos pelas instruções a cada instante de tempo. São utilizados registradores lógicos, cujas referências são convertidas a registradores de hardware no momento de sua utilização.

6ª Questão: Explique o que são, compare e exemplifique as seguintes técnicas: Delayed Branch e Otimização do Branch.

Delayed Branch:

Delayed Branch, é a técnica onde não se toma o desvio até que seja necessário, utilizamos de NOOP na técnica, pois é uma forma eficiente de não fazer nada e ele não atrapalha (além de perder tempo) o pipeline. É uma técnica mais fácil e segura em comparação com a otimização do Branch, pois mesmo que coloque mais Noop que o necessário, o programa só ficará mais lento e longo, o lado negativo desta técnica, é que não se ganha tempo ao utilizá-la.

Um exemplo de Delayed Branch x Normal:

Delayed	Normal:
1 Load X,Z	1 Load X,Z
2 Add 10, Z	2 Add 10, Z
3 Sub B,Z	3 Sub B,Z
4 Jump Z<B 7	4 Jump Z<B 7
5 Noop	5 Sub B,Z
6 Sub B,Z	6 Store Z,A
7 Store Z,A	

Nesse exemplo, podemos ver que no pipeline normal, a instrução 5 entraria mas seria desnecessária, pois não deve ser executada.

Otimização do Branch:

É uma técnica que se aproveita de como o pipeline funciona, quando um jmp vai vai ser executado, existem outras instruções que já estão prontas para serem executadas, então trocamos de lugar (a colocamos antes) para que enquanto o jmp seja executado, as outras instruções já estejam encaminhadas para serem executadas.

É consideravelmente mais difícil de ser implementado pois existem nuances como jumps condicionais que se realizados antes iriam atrapalhar o código. A grande vantagem é o ganho de tempo, pois em teoria não há perda de tempo.

Por exemplo:

Otimizado x Normal:

Otimização	Normal
1 Load R1, 6	1 Load R1, 6
2 Jump 5	2 Add 3,R1
3 Add 3,R1	3 Jump 5
4 Add 2,R1	4 Add 2,R1
5 Sub 1, R1	5 Sub 1, R1
6 Store R1	6 Store R1

Neste caso, aproveitamos que a instrução Add 3,R1 já está buscada e podemos a executar sem problemas após o jmp, diferente do normal em que a instrução add 2,r1 é desnecessária.