UNIVERSIDADE DE SÃO PAULO
ICMC - INSTITUTO DE CIÊNCIAS MATEMÁTICAS E COMPUTAÇÃO

DEPARTAMENTO DE CIÊNCIA DE COMPUTAÇÃO

Disciplina: SSC0510 Arquitetura de Computadores

Professora Eduardo do Valle Simões

Grupo 99

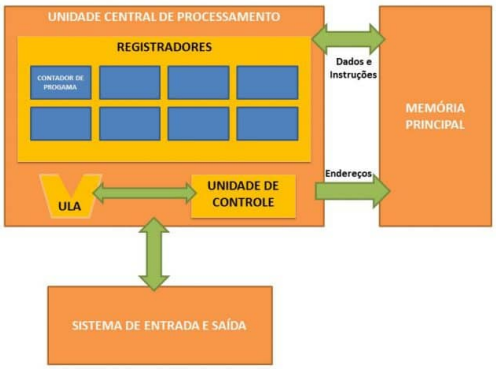
ADRIAN PEREIRA DA SILVA - N º USP: 11816171
JOÃO VITOR DIOGENES - Nº USP: 11816122
JULIO IGOR CASEMIRO OLIVEIRA - Nº USP: 11816139
MARCIO HIDEO ISHIKAWA - Nº USP: 11832902

Avaliação - Arquitetura de Computadores

São Carlos/SP
2021

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

O SISD, Single Instruction Single Data, é a classificação de Flynn que representa arquiteturas como de von Neumann. Possui somente um fluxo de instrução e de dados, permitindo uma operação por vez para ser executada, entretanto, é possível o uso de pipeline para agilizar o processo. Exemplo : Arquitetura Von Neumann



O SIMD, Single Instruction Multiple Data, representa os processadores matriciais, paralelos e associativos, ou seja, possibilita a execução de uma instrução em um conjunto de dados distintos simultaneamente. O SIMD é bastante presente nas placas de vídeo, pois uma mesma operação executada em uma imagem é realizada em outros conjuntos de dados.
Exemplo: Placa de vídeo

O MISD, Multiple Instruction Single Data, multiprocessadores que compartilham o mesmo conjunto de dados que executam operações diferentes. Não há um computador que utilize o MISD.
O MIMD, Multiple Instruction Multiple Data, são multiprocessadores que executam distintas instruções com conjunto de dados distintos. Um dos modelos mais utilizados nos dias atuais, pela sua eficiência, presente bastante nos notebooks e nos supercomputadores.
Exemplo: Intel Core i7

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

Uma arquitetura com memória compartilhada é quando uma mesma memória física é acessada por dois ou mais processadores, no qual a sincronização entre tarefas é feita por escrita/leitura na/da memória compartilhada e o usuário é o responsável pela sincronização. A comunicação entre tarefas é majoritariamente rápida, no entanto, uma região na memória não pode ser alterada por duas tarefas simultâneas, isso se deve para haver coerência de dados. Além disso, essa arquitetura possui escalabilidade limitada pelo número de barramentos conectando processadores e memória.

Uma arquitetura de memória distribuída consiste em múltiplos nós de processamento independentes com módulos de memória local, conectados por uma rede de comunicação, no qual os processadores que executam cada tarefa podem se comunicar e se manter sincronizados. Essa arquitetura possui uma boa escalabilidade, pois conseguem ser formadas por uma grande quantidade de máquinas. Por consequência, gera um alto custo de investimento e aumenta a complexidade de programação, mas simplifica a implementação do hardware, porque o sistema não precisa manter a ilusão de que toda a memória é realmente compartilhada.

Diferenças entre arquiteturas com memória compartilhada e memória distribuída.

- Memória Compartilhada (Multiprocessors)
    - Permite que vários elementos de processamento compartilhem o mesmo local na memória (ou seja, cada um lê e grava)
    - Hardware mais complexo
    - Programação mais simples
    - Escalabilidade limitada
- Memória Distribuída (Multicomputers)
    - Memória distribuída requer comandos explícitos para transferir dados de um elemento de processamento para outro.
    - Hardware mais simples
    - Programação mais complexa
    - Boa escalabilidade, pode ser expandido para muitos processadores

Um exemplo para a arquitetura de memória compartilhada é o **Cray T3E**

Um exemplo para a arquitetura de memória compartilhada é o **Sequent – 252 Pentium II**

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

1. **Processador com pipeline de operações**

O pipeline é semelhante ao uso de uma linha de montagem numa planta industrial. Uma linha de montagem tira a vantagem do fato de que um produto passa por vários estágios da produção dessa forma produtos em vários estágios podem ser trabalhados simultaneamente.

Portanto o pipeline acelera a execução de instruções, se seus estágios forem de durações iguais o ciclo de cada instrução será reduzido. No entanto, se não forem de tamanhos iguais haverá espera em vários estágios do pipeline. O desempenho do pipeline também é afetado pela instrução de desvio condicional, que pode invalidar várias leituras de instruções. O sistema deve conter uma lógica para lidar com esses e outros conflitos.

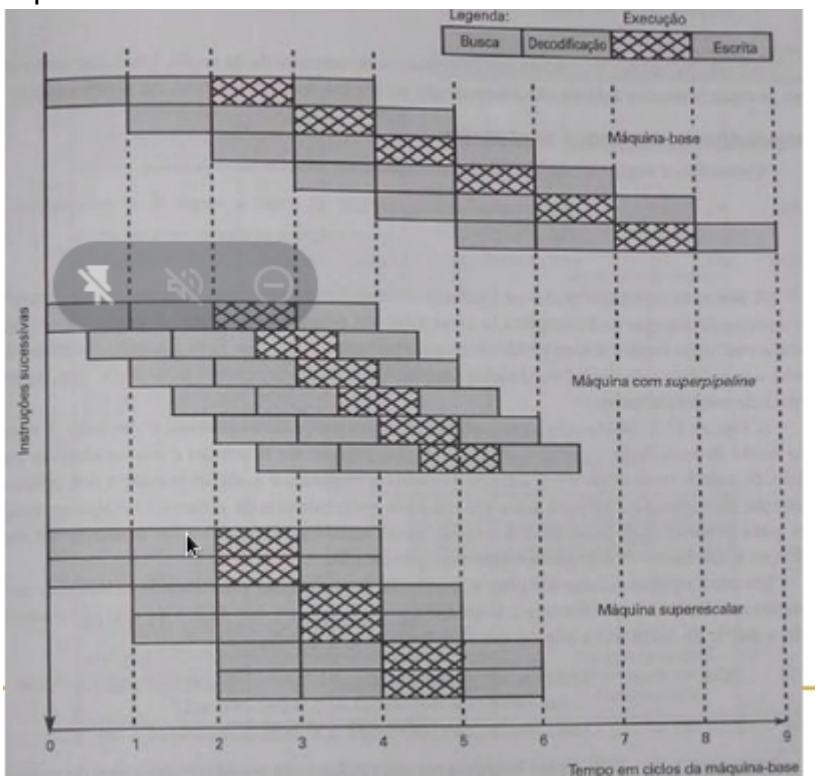
Inicialmente o pipeline estava presente apenas nos CPUs RISC porém agora são muito populares em todas as CPUs modernas. A quantidade de estágios de pipeline que Pentium II possui são 10, o Athlon Thunderbird possui 12 e o Pentium 4 possui 20 estágios.

2. **Processadores Superescalares**

Temos uma arquitetura superescalar quando as instruções comuns - como carregamento de valor da memória para registrador, armazenamento do registrador na memória, desvios condicionais - podem ser iniciadas simultaneamente e executadas de modo independente.

A essência da abordagem superescalar é a habilidade de executar instruções independente e concorrentemente em pipelines diferentes. O conceito pode ser ainda mais explorado permitindo que as instruções sejam executadas em uma ordem diferente da ordem do programa.

Temos o projeto superescalar atrelado a arquitetura RISC, as primeiras máquinas superescalares se tornaram disponíveis depois de um ou dois anos após a invenção do termo superescalar.



3. Processadores Paralelos

Sistemas multiprocessadores contêm dois ou mais processadores com capacidades aproximadamente idênticas, compartilhando módulos de memória, canais de entrada/saída e dispositivos periféricos. O sistema multiprocessador deve ser controlado por um único sistema operacional, controlando as interações entre processadores e programas. Cada processador possui a sua memória local e dispositivos próprios, sendo que a comunicação entre processadores se faz através das memórias compartilhadas ou de uma rede de interrupção.

Como exemplo para máquinas desse tipo podemos citar o Nintendo DS que possui dois processadores o ARM7TDMI e ARM946E-S.

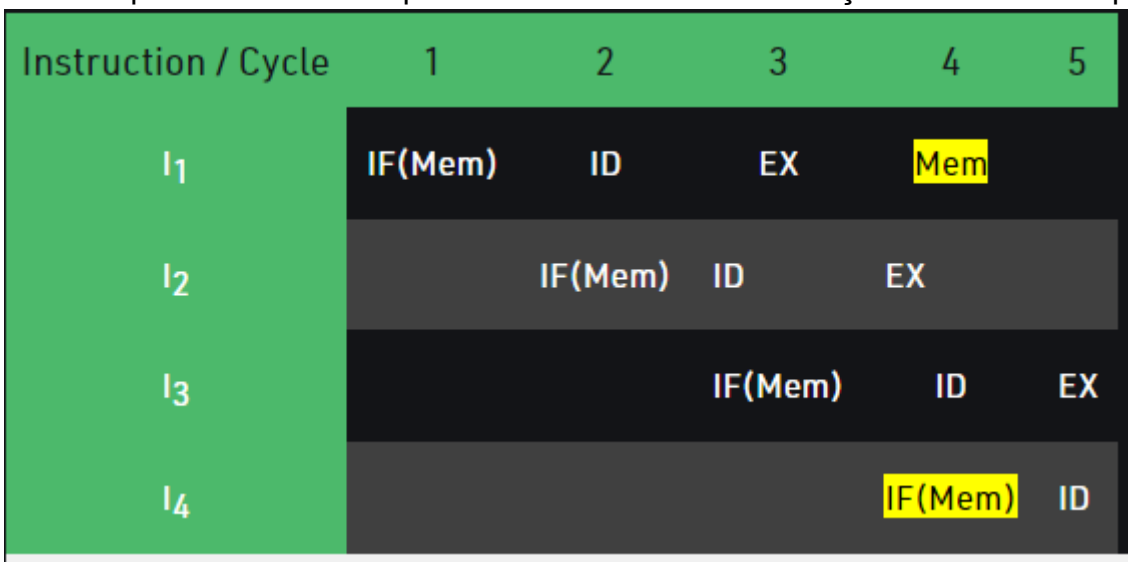
4a Questão: Explique as limitações intrínsecas do paralelismo: Dependência de dados, Dependência de desvio, Conflito de recurso (ULA) e o que pode ser feito para minimizar esses problemas.

As limitações intrínsecas do paralelismo resumem-se à ideia de que muitos programas são inerentemente sequenciais e exigem determinada ordem na execução das instruções. Sendo assim, têm-se os conceitos de dependência de dados, dependência de desvio e conflito de recurso.

Dependência de dados trata-se de uma instrução que depende dos dados de alguma instrução anterior, por exemplo, a instrução “ADD R3, R3, R2”, depende de qualquer instrução que altere o valor dos registradores R3 e R2 que estejam antes dela.
A solução nesse caso trata-se do controle dos registradores e a técnica de renomeação de registradores, por exemplo.

A dependência de desvio, de forma semelhante, trata-se de uma instrução de desvio condicional que depende de algum dado modificado em instrução anterior, de forma que a ordem não possa ser alterada sem alterar o comportamento do programa, ou também, no caso em que um desvio acontecerá e não podemos executar mais instruções (em um pipeline normal), causando “stall” em run-time.
A solução para essa limitação são as técnicas de otimização de pipeline e de branch-prediction.

Já o conflito de recurso diz respeito aos componentes de hardware disponíveis e ao fato de que eles não podem ser usados por mais de um bloco de instrução ao mesmo tempo. Por exemplo:



A execução I1 e I4 utilizam o mesmo recurso ao mesmo tempo (ambas acessam a memória no tempo 4), o que seria um problema e jamais pode ocorrer. Apenas uma tarefa pode utilizar um componente da arquitetura por vez.
Podemos minimizar esse problema com arquiteturas mais robustas, como por exemplo, arquiteturas de harvard que separam memórias de programa e memórias de dados, ou também com componentização mais específica, fazendo com que as instruções usem recursos mais diferentes, como por exemplo, um processador de incremento para evitar que operações vetoriais tenham de usar a ULA tão frequentemente.

5a Questão: Explique renomeação de registradores.

Os programas são compostos por instruções e operam sobre valores alocados em registradores. No entanto, as instruções não demoram o mesmo tempo para serem executadas e ao acabarem de executar, liberam o processador para executarem outras instruções, que, por sua vez, acessam novamente os registradores. Entretanto, como forma de melhorar o desempenho do computador, as instruções poderão ser executadas em qualquer ordem, observando as unidades de execução especializadas.

E quando há essa troca de ordem, podem haver conflitos quando há dependências entre instruções de um programa, imposta pelo reuso dos registradores por essas operações. E a Renomeação de Registradores refere-se a uma técnica utilizada para eliminar essas dependências e aumentar o paralelismo disponível, permitindo que duas instruções que eram executadas serialmente sejam executadas concorrentemente já que estão utilizando registradores diferentes.

Original

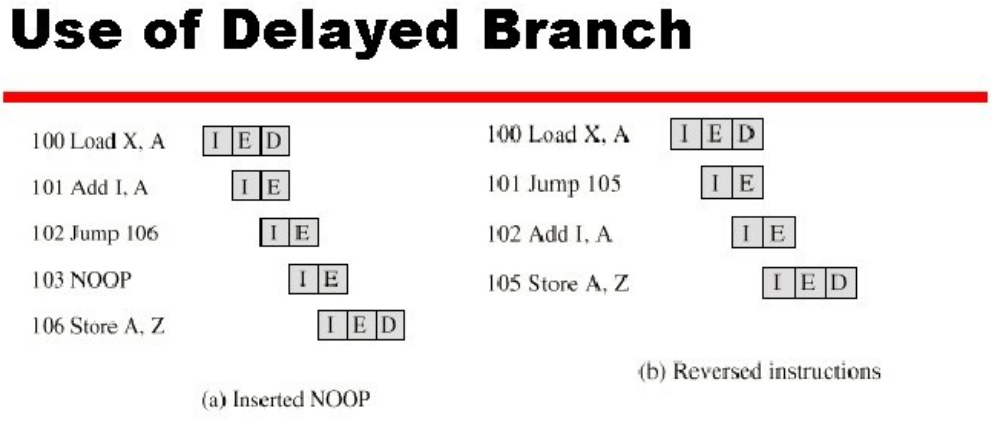
R1 = 5
MEN[0] = R1
R1 = 10
MEN[1] = R1

Renomeado

R1 = 5
MEN[0] = R1
R9 = 10
MEN[1] = R9

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

Delayed Branch e otimização do Branch são técnicas de otimização de pipeline relacionadas à forma de lidar com branchs, que são ramificações, ou de forma mais comum, desvios dentro da execução de um programa.
Desvios são os piores inimigos de pipelines porque fazem com que todos os estágios já feitos para determinada instrução sejam descartados caso um desvio seja feito no estágio de execução, visto que será necessário buscar uma instrução totalmente nova, perdendo vários ciclos do processador. Também é necessário, no acontecimento de desvios, que hajam formas de controle, para que os carregamentos das instruções anteriores que não serão executados não sejam dados residuais que atrapalhariam as próximas instruções., seja com uma máquina de controle muito elaborada e complexa, ou com técnicas como estas.
A técnica de Delayed Branch trata-se de literalmente inserir “x - 1” (sendo x o número de estágios da máquina) NOOPs depois de cada instrução de desvio (lembrando de também adicionar 1 ao endereço do jump). Essa tarefa é feita pelo compilador e faz com que o hardware seja simplificado, sem precisar se preocupar com o tratamento dos desvios. Não é necessário limpar nenhum dado, visto que apenas NOOPs foram executados após o início da instrução de desvio e mantemos a máquina de controle mais simplificada. O grande problema é que perderemos ciclos da mesma forma.
Já a técnica de Optimized Branch trata-se de alterar a ordem de execução das instruções. De forma simplificada, atrasamos “x - 1” instruções (sendo x a quantia de estágios da máquina) para serem executadas após o Jump, de forma que leiamos o jump e em seguida as operações que estavam anteriores a ele. Assim, acontece o seguinte comportamento mostrado nos exemplos, de forma que não percamos ciclos de execução:



ADDRESS	(a) NORMAL BRANCH		(b) DELAYED BRANCH		(c) OPTIMIZED DELAYED BRANCH	
100	load	*X, A*	load	*X, A*	load	*X, A*
101	add	*1, A*	add	*1, A*	jump	105
102	jump	105	jump	106	add	*1, A*
103	add	*A, B*	nop		add	*A, B*
104	sub	*C, B*	add	*A, B*	sub	*C, B*
105	store	*A, Z*	sub	*C, B*	store	*A, Z*
106			store	*A, Z*		

O grande problema nessa técnica se deve aos conceitos de dependência, visto que algumas instruções dependem de valores em registradores que ainda não foram executados, de forma que a ordem das instruções não possa ser muito alterada e NOOPs tenham de ser adicionados de qualquer forma. Um exemplo é quando temos um Jump condicional e a instrução que resulta o valor a ser analisado pelo Jump é a instrução imediatamente anterior ao Jump.