Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação e Estatística

Disciplina SCE541
Arquitetura de Computadores

Grupo nr.: 42

Nome: Altair Fernando Pereira Junior Nº USP: 9391831
Nome: Guilherme Holanda Sanches Nº USP: 10734370
Nome: Ygor Pontelo Nº USP: 10295631

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

R. Single Instruction Single Data (SISD), também conhecida por ser uma das arquiteturas mais simples, possui uma única unidade de controle podendo possuir mais de uma unidade funcional. Opera de modo sequencial característico da máquina de Von Neumann, sendo assim no SISD um único fluxo de instruções opera sobre um único fluxo de dados.

Single Instruction Multiple Data (SIMD), ao contrário do SISD que opera um único dado por instrução, o SIMD processa de vários dados sob o comando de apenas uma instrução. Ainda opera de modo sequencial, mas para possibilitar o acesso a múltiplos dados é preciso uma organização de memória em diversos módulos. A unidade de controle é única e existem diversas unidades funcionais

Multiple Instruction Single Data (MISD) é um arquitetura onde que ao contrário do SISD e do SIMD que operam sob o comando de uma instrução em um único dado, ele possui múltiplas unidades de controle executando instruções distintas operando sobre o mesmo dado.

Multiple Instruction Multiple Data (MIMD) ao contrário das anteriores possuio processamento de múltiplos dados por parte de múltiplas instruções. Neste caso, várias unidades de controle comandam suas unidades funcionais, as quais têm acesso a vários módulos de memória. Isso ocorre em arquiteturas paralelas, servidores, etc.

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

R. Arquiteturas com memória compartilhada possuem um mesmo conjunto de dados que pode ser operado por processadores diferentes no mesmo computador. Ou seja, pode haver um processo com múltiplas threads que possuem a área de instruções, memória, arquivos e dispositivos de I/O compartilhados e trabalham juntas para realizar as tarefas. Nem tudo é compartilhado por essas threads, cada uma deve ter o seu Program Counter específico para realizarem rotinas diferentes, e muitas vezes pode haver competição para acesso a recursos, então o programador precisa utilizar artefatos como o Mutex, semáforos ou a arquitetura de monitor para garantir a integridade dos dados compartilhados, evitar os deadlocks, onde múltiplas threads co-dependem de recursos que já estão alocados para seguir com a sua execução.

Arquiteturas com memória distribuída, ou sistemas distribuídos são compostos por múltiplos computadores formando clusters que compartilham a memória através de mensagens (http, sockets) em uma rede. Este tipo de arquitetura garante às tarefas grande disponibilidade, uma vez que em caso de falha, um computador pode receber requisições no lugar de outro e geralmente oferecem grande escalabilidade, por isso atualmente essas arquiteturas são utilizadas nos serviços das grandes empresas de tecnologia, como o Facebook, Google, Twitter.

Em questão de velocidade de trocas de informações e facilidade de programação, arquiteturas com memória compartilhada saem na frente pois cada tarefa compartilha informação quase instantâneamente, e o gerenciamento dos recursos é feito em grande parte pelo sistema operacional. Agora levando em conta problemas com grande quantidade de dados, sistemas distribuídos podem ser mais baratos. Um exemplo é o armazenamento e distribuição de dados por meio de bancos de dados não relacionais,

múltiplos computadores em conjunto interagem com uma quantidade massiva de informações, um computador com tal capacidade de armazenamento e processamento seria muito caro e não traria a mesma disponibilidade para o serviço.

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

1. Processador com pipeline de operações
2. Processadores Superrescalares
3. Processadores Paralelos

R. Um processador com pipeline de operações tira proveito do fato de que a execução de uma instrução é naturalmente dividida, pelo menos, na fase de busca da instrução, decodificação e execução da instrução, e em muitas vezes há também a fase de escrita na memória como nas instruções de load e store. Estas fases podem ser sobrepostas, executando diversos ciclos ao mesmo tempo, pois não utilizam os mesmos recursos do processador. Atualmente a maioria dos processadores funcionam dessa maneira pois a velocidade de processamento aumenta a cada subdivisão de estágios do pipeline.

Já os processadores superescalares são os que possuem múltiplas unidades de execução com pipelines paralelos para uma stream de instruções, portanto podem realizar mais de um ciclo de busca, decodificação, execução e acesso à memória em um mesmo ciclo de clock. Para isso adotam estratégias para a minimização das dependências de dados, de desvio e de competição por recursos (que serão explicadas na próxima questão) entre as instruções para tirar o maior proveito da execução paralela.

Existem também as máquinas que possuem múltiplos processadores que são controlados por diferentes unidades de controle. Esses processadores podem executar diferentes streams de instruções ao mesmo tempo, sendo elas processos diferentes ou múltiplas threads de um mesmo processo. Isso não exclui a possibilidade de que cada um desses núcleos de processamento possam ter uma arquitetura superescalar.

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.

R. Na dependência de dados uma instrução utiliza um operando que é produzido para uma instrução anterior

ADD r1, r2 (r1 := r1+r2;)
MOVE r3,r1(r3 := r1;)

Mas não pode executar a segunda instrução até que a primeira tenha terminado. Ou seja, uma instrução deve ser atrasada até que todos os dados sejam produzidos

Na dependência de não se pode executar instruções depois de um desvio em paralelo com as instruções antes do desvio. Assim, como o tamanho da instrução não é conhecido, uma instrução deve ser decodificada, pelo menos parcialmente, antes que a instrução seguinte possa ser buscada. Isso impede a busca simultânea de instruções e consequentemente impede o conflito.

No conflito de recursos duas ou mais instruções que necessitam usar o mesmo recurso ao mesmo tempo

- •Acesso concorrente à memória
- •Acesso concorrente a um banco de registradores
- •Uso simultâneo de unidades funcionais

Sendo assim os conflitos de recursos podem ser superados pela duplicação de recursos, enquanto uma dependência de dados não pode ser eliminada. Além disso, quando uma operação efetuada em uma dada unidade funcional consome muito tempo para ser completada, é possível minimizar os conflitos de uso dessa unidade por meio de sua implementação como uma pipeline.

5ª Questão: Explique renomeação de registradores.

R. Renomeação de registradores é uma técnica para explorar paralelismo e possibilitar que o processador consiga executar mais instruções ao mesmo tempo. Para isso, o programador usa apenas os registradores da arquitetura, o que remove as dependências de dados falsa (Antidependência e Dependência de Saída). Sendo assim, é possível identificar as dependências verdadeiras e otimizar a execução das instruções, mesmo que seja fora de ordem, aumentando o desempenho do programa.

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

R.

Delayed Branch: Basicamente colocamos a instrução NOOP depois de uma instrução de desvio condicional, isso ajuda pois elimina a necessidade de ter uma máquina de controle para gerenciar a limpeza do pipeline durante um desvio. A razão é que não sabemos o resultado da condição até o próximo ciclo (pois primeiro temos que buscar a instrução), para não perder ciclos, buscamos a próxima instrução independente do desvio ocorrer ou não, que no caso do NOOP seria uma instrução que não faz nada e não prejudica o algoritmo.

Otimização do Branch: Como o nome sugere, tenta otimizar as instruções. Pegando o exemplo do pdf (Ver abaixo), ao invés de colocarmos o NOOP, colocamos a instrução de desvio primeiro e então colocamos a instrução de soma, pois esta tem que ser executada de qualquer maneira. Nesse exemplo evitamos colocar uma instrução a mais que não faria nada e ainda não precisando de uma máquina de controle.

As duas são importantes, mas é preferível usar a Otimização de Branch quando possível pois não aumenta o número de instruções.

Otimização do Pipeline

Address	Normal	Delayed	Optimized
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	NOOP	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	