Unidade 12: Introdução ao Paralelismo:

Processadores Superescalares

Prof. Daniel Caetano

Objetivo: Apresentar os conceitos fundamentais da arquitetura superescalar e as bases do paralelismo de processamento.

Bibliografia:

- STALLINGS, W. **Arquitetura e organização de computadores**. 5ed. São Paulo: Ed. Pearson Prentice Hall, 2003.
- MURDOCCA, M. J; HEURING, V.P. **Introdução à arquitetura de computadores**. S.I.: Ed. Campus, 2000.

INTRODUÇÃO

Na aula anterior foi apresentada uma forma de organização das partes de um processador chamada "pipeline", que permitia que praticamente todos os elementos da CPU estivessem funcionando ao mesmo tempo. Apesar de acelerar bastante o processamento (em especial se o código foi otimizado para isso), o pipeline ainda representa uma forma sequêncial de processamento.

Na aula de hoje começaremos a estudar o processamento paralelo, conhecendo seus fundamentos iniciais. Iniciaremos com a arquitetura superescalar, que foi introduzida com o PowerPC e o Pentium II e, em seguida, serão apresentados os fundamentos do multiprocessamento.

1. ARQUITETURA SUPERPIPELINE

Na aula passada foi apresentado o conceito de "pipeline", isto é, uma forma de organizar o funcionamento da CPU para que ela seja capaz de processar as instruções mais rapidamente, minimizando a ocorrência de circuitos ociosos.

	SEM pipeline		COM pipeline	
Seqüência no Tempo	Busca	Execução	Busca	Execução
0	I1	-	I1	-
1	-	I1	I2	I1
2	I2	-	I3	I2
3	-	I2	I4	I3
4	I3	-	I5	I4

Alguns pesquisadores, entretanto, perceberam que, quando se subdividia a pipeline em um maior número de níveis, alguns destes estágios precisavam de muito menos tempo de execução que um ciclo de clock.

	SEM Pipeline						COM Pipeline					
T	BI	DI	CO	BO	EI	EO	BI	DI	CO	BO	EI	EO
0	I1	-	-	-	-	-	I1	-	-	-	-	-
1	-	I1	-	-	-	-	I2	I1	-	-	-	-
2	-	-	I1	-	-	-	I3	I2	I1	-	-	-
3	-	-	-	I1	-	-	I4	I3	I2	I1	-	-
4	-	-	-	-	I1	-	I5	I4	I3	I2	I1	-
5	-	-	-	-	-	I1	I6	I5	I4	I3	I2	I1
6	I2	-	-	-	-	-	I7	I6	I5	I4	I3	I2
7	-	I2	-	-	-	-	I8	I7	I6	I5	I4	I3
8	-	-	I2	-	-	-	I9	I8	I7	I6	I5	I4
9	-	-	-	I2	-	-	I10	I9	I8	I7	I6	I5
10	-	-	-	-	I2	-	I11	I10	I9	I8	I7	I6
11	-	-	-	-	-	I2	I12	I11	I10	I9	I8	I7
12	I3	-	-	-	-	-	I13	I12	I11	I10	I9	I8

Assim, propôs-se o uso de um duplicador (ou até triplicador) de clock interno na CPU, de maneira que muitos destes estágios do pipeline pudessem ser finalizados na metade do tempo, acelerando ainda mais o processamento final.

É importante notar, porém, que o aumento de desempenho é oriundo de um menor tempo de processamento de cada instrução, mas elas ainda estão sendo executadas uma a uma.

2. ARQUITETURA SUPERESCALAR

A Pipeline, que chegou aos computadores caseiros com o Pentium, trouxe um grande domínio da Intel no mercado de computadores domésticos, mas a empresa sentia que precisava ampliar ainda mais a capacidade de seus processadores, que vinham sendo seguidos de perto pelos de sua principal concorrente nesse segmento: a Advanced Micro Devices, AMD.

Expandindo a idéia de processar simultaneamente os diferentes estágios de insturções diferentes, foi sugerida a implementação de várias pipelines para a CPU, de maneira que várias instruções pudessem, de fato, serem executadas ao mesmo tempo, desde que fossem independentes. O princípio é o mesmo da pipeline já visto, mas nessas novas CPUs foram inseridas, em uma mesma CPU, diversas pipelines.

Essa estratégia é conhecida como **paralelismo em nível de instruções**, porque ela é capaz de detectar instruções independentes e executá-las simultaneamente. Em outras palavras, a Unidade de Controle, ao ler instruções, vai "jogando-as" para os diferentes

pipelines, alternadamente. Desta maneira, diversas instruções são processadas em um intervalo de tempo que só seria possível processar uma instrução, caso existisse um único pipeline.

2.1. O que São Instruções Independentes?

Talvez seja mais mais simples dizer o que são **instruções dependentes**. Instruções dependentes são aquelas cujo resultado de uma depende do resultado de outra e, assim, a ordem de execução deve ser preservada. Por exemplo:

```
LD    A,17
ADD   A,20
```

Resulta no valor 37 armazenado no registrador A. Entretanto, para que a instrução ADD A,20 possa ser executada, a instrução LD A,17 já precisa ter terminado.

Esse tipo de problema, como já foi comentado, atrapalha até mesmo o pipeline simples, mas vira um problema muito sério quando se usa uma arquitetura superescalar, pois o uso excessivo deste tipo de instruções acaba desperdiçando desempenho da CPU.

Como é impossível evitar essa dependência em muitos casos, a solução são os "compiladores otimizadores" ou mesmo o uso de Unidades de Controle mais inteligentes. O que eles fazem? Simples: analisam e **mudam a ordem do código** de acordo com as dependências. Por exemplo:

```
LD    A,17        ; Armazena 17 em A
ADD   A,20        ; Soma 20 em A
LD    C,A         ; Guarda resultado (37) em C
LD    A,30        ; Armazena 30 em A
ADD   A,10        ; Soma 10 em A
LD    D,A         ; Armazena o valor de A em D
```

Neste código temos dois blocos "quase" independentes:

```
LD    A,17        ; Armazena 17 em A
ADD   A,20        ; Soma 20 em A
LD    C,A         ; Guarda resultado (37) em C

LD    A,30        ; Armazena 30 em A
ADD   A,10        ; Soma 10 em A
LD    D,A         ; Armazena o valor de A em D
```

Estes dois blocos poderiam ser paralelizados trocando-se o registrador A do segundo bloco por outro registrador (se isso for possível), como o registrador B:

```
LD    A,17      ; Armazena 17 em A
ADD   A,20      ; Soma 20 em A
LD    C,A       ; Guarda resultado (37) em C
LD    B,30      ; Armazena 30 em B
ADD   B,10      ; Soma 10 em B
LD    D,B       ; Armazena o valor de B em D
```

Agora os blocos são totalmente independentes, embora as insutrções em cada um deles ainda precise ser executada na sequência. Como otimizar isso para o uso de arquitetura superescalar? Simples: vamos intercalar os blocos!

```
LD    A,17      ; Armazena 17 em A
LD    B,30      ; Armazena 30 em B
ADD   A,20      ; Soma 20 em A
ADD   B,10      ; Soma 10 em B
LD    C,A       ; Guarda resultado (37) em C
LD    D,B       ; Armazena o valor de B em D
```

Agora essas instruções podem ser processadas com otimalidade em uma arquitetura com duas pipelines!

Existem situações, entretanto, que a dependência é mais complexa, como as que envolvem deslocamento de execução, com JMP e, especialmente, com CALL. O CALL é uma forma de executar uma subrotina, o que significa que, quando se executa um CALL, em algum momento ocorrerá um RETurn. Ocorre que a instrução RET pode ter comportamento variado, dependendo do resultado de uma comparação. Por exemplo:

```
CP    A,B
RET   Z
```

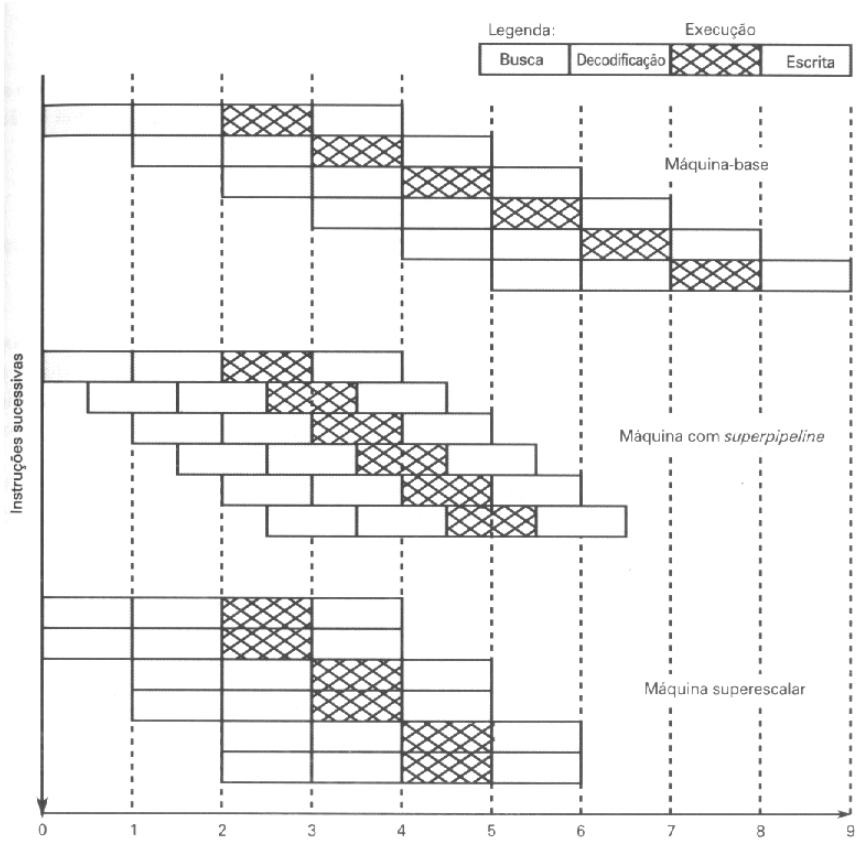
Esse RETurn Z indica que só haverá o retorno caso A e B sejam iguais. Esse tipo de dependência nem sempre é fácil de resolver, mas os bons compiladores e processadores são capazes de usar artifícios para contornar estas situações.

3. COMPARAÇÃO: PIPELINE x SUPERPIPELINE x SUPERESCALAR

O gráfico a seguir mostra o processamento superescalar comparado com o processamento em pipeline e superpipeline. Observe que, diferentemente das arquiteturas pipeline e superpipeline, na arquitetura superescalar temos, de fato, mais de uma instrução sendo processada ao mesmo tempo.

Isso é muito útil para processamento dados como imagens e outros tipos de matrizes, onde é possível processar diversos dados diferentes ao mesmo tempo pois, em geral, o

processamento de cada pixel ou elemento da matriz é independente do processamento dos outros pixels/elementos da matriz.



3.1. Lógica de Execução SuperEscalar

A sequência que o processador segue para executar um programa superescalar é indicada na figura a seguir:

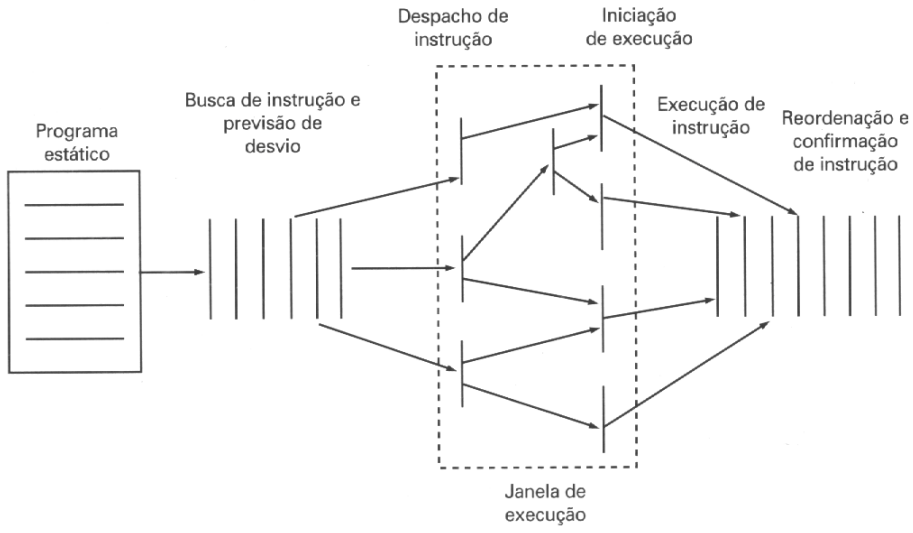


Figura 13.6 Representação conceitual de processamento superescalar (Smith e Sohi, 1995).

4. FUNDAMENTOS DO MULTIPROCESSAMENTO

O processamento escalar permite um certo nível de processamento paralelo, chamado "paralelismo em nível de instruções", como já citado. Entretanto, a idéia de execução de múltiplas tarefas simultaneamente pode ser expandida ainda mais.

De fato, o conceito de processamento paralelo é usado, atualmente, para se referir ao "paralelismo em nível de processos", em que diferentes processos são executados simultaneamente.

Existem três tipos básicos de processamento paralelo em uso corrente na atualidade:

SMP - Symetric MultiProcessing: multiprocessamento simétrico, em que vários processadores compartilham a mesma infraestrutura de computador (dispositivos e memória).

Clusters - multiprocessamento usando vários computadores de forma que eles todos trabalhem como se fossem apenas um.

NUMA - Non-Uniform Memory Access: acesso não uniforme à memória, em que vários processadores compartilham a mesma infraestrutura de computador, mas cada um deles trabalhando, em geral, em uma região específica da memória, diferente para cada um deles.

Na próxima aula veremos um pouco mais de detalhes sobre cada um desses, no momento iremos entender a taxonomia dessas formas de processamento paralelo, proposta por Flynn (1972 *apud* Stallings, 2003):

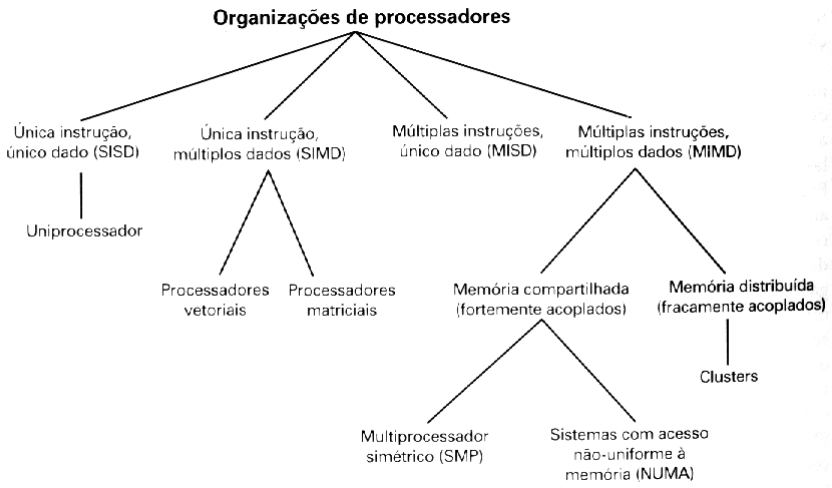
SISD - Single Instruction, Single Data (Única Instrução, Único Dado): sistemas monoprocessados. Pipelines e processadores escalares entram nessa categoria.

SIMD - Single Instruction, Multiple Data (Única Instrução, Múltiplos Dados): sistemas de processamento vetorial e matricial.

MISD - Multiple Instructions, Single Data (Múltiplas Instruções, Único Dado): seria um sistema em que diferentes conjuntos de instruções se aplicassem ao mesmo dado, simultaneamente. É um conceito teórico.

MIMD - Multiple Instructions, Multiple Data (Múltiplas Instruções, Múltiplos Dados): várias unidades de processamento, processando diferentes informações, como no caso dos sistemas SMP e clusters.

A figura a seguir esquematiza a relação entre as classificações e diferentes sistemas. Cada uma dessas formas de multiprocessamento será descrita na próxima aula, mas a maioria delas já foi brevemente apresentada, com exceção do processamento vetorial/matricial. O **processamento vetorial** se refere a sistemas que precisam executar uma única instrução sobre um grande conjunto de dados como, por exemplo, o processamento de uma imagem ou a simulação de partículas. Os computadores de processamento vetorial são frequentemente chamados de **supercomputadores**.



5. O CONCEITO DO COMPUTADOR COMPLETO (WHOLE COMPUTER)

Tanto os sistemas SMP/NUMA quando os Clusters são sistemas com uma arquitetura MIMD, isto é, são capazes de processar múltiplos dados com instruções diferentes de maneira simultânea. A diferença mais comum entre eles fica explícita pelo conceito de **computador completo**.

Um **computador completo** é uma unidade de processamento totalmente funcional, isto é, com sua própria memória, dispositivos e cpu. Os sistemas **SMP** e **NUMA** possuem multiprocessamento mas não são compostos por vários computadores completos, já que uma grande parte dos recursos de cada uma das unidades de processamento é compartilhado com as demais.

Já os **Clusters** são, em geral, compostos pela união de vários computadores completos, que trabalham em conjunto como se fossem um único computador. Cada um dos computadores completos que compõem um cluster recebe o nome de **nó**. Assim, cada nó possui sua própria memória e seus dispositivos, sendo uma unidade de processamento independente.

Resumidamente, um cluster com **N** nós pode ser separado em **N** computadores independentes. Um sistema SMP ou NUMA com **N** unidades de processamento não pode ser dividido em **N** computadores independentes.

6. BIBLIOGRAFIA

STALLINGS, W. **Arquitetura e organização de computadores**. 5ed. São Paulo: Ed. Pearson Prentice Hall, 2003.
MURDOCCA, M. J; HEURING, V.P. **Introdução à Arquitetura de Computadores**. S.I.: Ed. Campus, 2000.