ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES

Aline Zanin

Funcionamento e soluções (*pipeline*)

Objetivos de aprendizagem

Ao final deste texto, você deve apresentar os seguintes aprendizados:

- Descrever o conceito de *pipeline*.
- Identificar as características dos *pipelines* de instruções.
- Reconhecer os tipos de *pipeline*.

Introdução

Podemos dizer analogamente que o processador — unidade central de processamento (UCP, ou CPU, do inglês *central processing unit*) — é o coração do computador. Nenhuma atividade é executada sem passar por ele. Em alguns casos, as instruções que precisam ser processadas pela CPU são complexas e demandam tempo. Para reduzir esse tempo, são utilizadas estratégias de paralelismo.

Existem dois tipos principais de paralelismo. O paralelismo em nível de *hardware é* alcançado com a replicação do número de unidades de processador, de modo que eles trabalhem em paralelo. Já o paralelismo em nível de instruções, conhecido como ILP (do inglês *instruction level parallelism*), trabalha com a organização das unidades do processador, para que não fiquem ociosas.

Há duas formas principais de implementar o ILP: uma delas é por meio do *pipeline*, e a outra é por meio de processadores superescalares. *Pipeline* é um conceito empregado pelos processadores no processamento de tarefas. Com ele, o processador paraleliza as atividades que precisam ser executadas, fazendo diversas atividades ao mesmo tempo, otimizando o trabalho e, reduzindo o tempo demandado para essas atividades.

Assim, neste capítulo, você vai estudar sobre o *pipeline*, verificando os tipos existentes e as características principais de um *pipeline* de instruções.

O que é *pipeline*

A busca por otimizar o processamento do computador, paralelizando a execução das instruções, não é recente. Desde o IBM Stretch (1959), os computadores já tinham a capacidade de buscar instruções em memória e armazená-las em um *buffer*, o que permitia que essas instruções fossem acessadas antecipadamente, isto é, antes da conclusão da leitura da memória do computador. Esse processo de busca antecipada, na verdade, paralelizava o processo, porque o dividia em duas partes, sendo elas a busca e a execução propriamente dita das instruções. O conceito de ***pipeline*** amplia essa estratégia, porque permite dividir uma instrução em diversas partes e dedicar uma parte do *hardware* para a execução de cada divisão da instrução, conforme Tanenbaum (2007).

O conceito de *pipeline* pode ser aplicado em situações cotidianas, e não apenas no contexto dos processadores de computador. Um exemplo claro é uma linha de produção, seja de carros e hambúrgueres. Imagine o seguinte cenário: em uma lanchonete, há cerca de 30 clientes na fila, e apenas um funcionário trabalhando. Esse funcionário executa sequencialmente as ações de: assar o pão, fritar o hambúrguer, cortar o pão, passar o molho no pão, fritar as batatas e, finalmente, montar o sanduíche.

É notório que esse processo seria otimizado caso essas ações fossem executadas paralelamente: enquanto o pão está assando, o hambúrguer pode ser assado; enquanto isso as batatas são preparadas; o pão é cortado e recebe o molho O mesmo acontece no computador: se o processador executar todas as tarefas sequencialmente, precisará de mais tempo. Se executá-las em paralelo, o tempo necessário será menor, aponta Tanenbaum (2007).

Todo processo que faz uso de *pipeline*, seja ele um *software* ou não, considera duas premissas básicas:

1. O processo é dividido em etapas independentes umas das outras.
2. Um novo produto inicia sua produção antes que o produto anterior tenha sido concluído.

O *pipeline* não reduz o tempo gasto para completar cada instrução individualmente — o ganho de tempo está no processamento simultâneo de diversas instruções, e não na velocidade de processamento individual de cada instrução. Esse processo de executar diversas instruções simultaneamente é chamado de paralelismo de instruções e tem por objetivo aumentar o número de execuções realizadas a cada ciclo do relógio.

Fique atento

Existe uma metodologia ágil de desenvolvimento de *software* que prega que tudo o que é bom deve ser explorado ao extremo (*extreme programming*, XP). No caso da arquitetura de processadores, se um *pipeline* é bom, por que não colocar dois, três ou mesmo quatro? Quando se replicam os *pipelines*, constrói-se o que chamamos de **arquitetura superescalar**.

Pipeline de instruções

As ações de processamento executadas pelo processador de um computador são conhecidas como instruções. Na seção anterior, vimos que um *pipeline* existe para paralelizar um trabalho, fazendo com que a sua execução deste em pequenas partes seja mais eficiente e mais rápida.

Um ***pipeline*** de instruções nada mais é do que a aplicação desse conceito nas instruções recebidas pelo processador. Ou seja, ao receber uma determinada instrução para ser processada, o processador vai dividi-la em etapas conforme o seu tipo e tamanho, para que sejam dedicadas partes do *hardware* para cada pedaço da instrução.

Saiba mais

Opcode (do inglês *operation code*) ou código de operação é um código fundamental para o funcionamento dos processadores, que é gerado em todas as operações executadas. Esse código é um intermédio entre a linguagem humana e a linguagem binário interpretada pelo processador.

Antes de tratarmos do paralelismo de instruções, vamos ver um pouco do ciclo natural que uma instrução tem em seu processamento. A execução de uma instrução é naturalmente dividida nas etapas de busca-decodifica e executa. Esse ciclo acontece da seguinte maneira: a CPU busca uma instrução, transferindo-a da memória principal para o registrador de instruções; poste-

riormente, essa informação é decodificada, ou seja, é determinado o *opcode*, e são carregados os dados necessários para a execução da instrução; por fim, a instrução é executada, com todas as suas operações.

Considere uma CPU com capacidade de execução de *pipeline*. Essa CPU pode receber uma instrução e quebrar esse processo em alguns "minipassos"; por exemplo:

1. buscar instrução;
2. decodificar *opcode*;
3. calcular endereço efetivo dos operandos;
4. carregar operandos;
5. executar instrução;
6. armazenar resultado.

Todos os estágios de um *pipeline* são conectados por uma estrutura chamada de duto, de forma que a instrução possa entrar em um lado do *pipeline* e sair concluída no outro lado. Na parte superior da Figura 1, temos um *pipeline* com cinco unidades; cada unidade está executando uma parte de uma tarefa relacionada à leitura e ao armazenamento de dados no computador. Na parte inferior da figura, podemos ver como esse *pipeline* funciona em função do tempo: a cada ciclo do relógio, um novo estágio é executado em paralelo com o estágio anterior, que já estava em execução. É possível que uma nova tarefa se inicie sempre sem a conclusão da tarefa anterior e que diversas tarefas sejam executadas ao mesmo tempo.

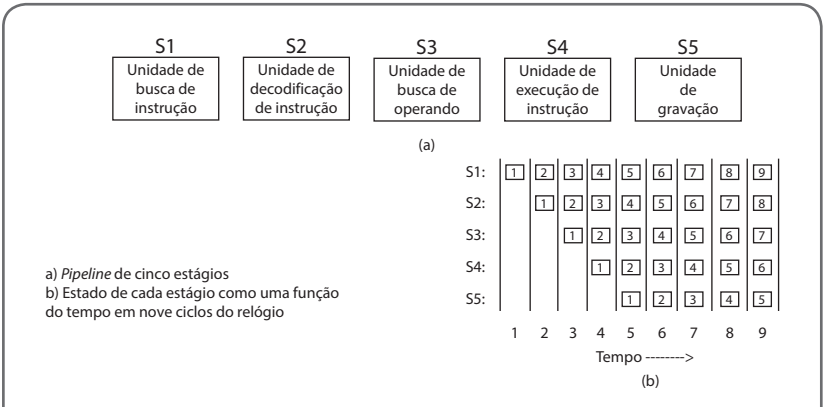

Figura 1. Ciclo de *pipeline*.

Fonte: Tanenbaum (2007, p. 35).

Um *pipeline* de instruções pode gerar dois problemas de conflito: conflito de recursos e dependência de dados. O conflito de recursos acontece quando duas ou mais instruções precisam acessar um recurso do computador simultaneamente. Por exemplo: quando uma instrução está armazenando um valor na memória, e, ao mesmo tempo, outra instrução está buscando valores na memória, ocorre um conflito de recursos, porque ambas necessitam acessar a memória. Geralmente, isso é resolvido ao se permitir que a instrução em execução prossiga, forçando a nova instrução a esperar, para depois cumprir o seu papel. Já a dependência de dados acontece quando uma instrução depende dos dados resultantes de outra instrução para poder ser executada ou para concluir o seu objetivo. Esse conflito, em geral, é resolvido com um tratamento de *hardware*, que vai identificar um possível conflito e retardar a execução da instrução dependente.

Link

Na animação disponível no link abaixo, você encontra uma explicação bastante didática sobre *pipelines* e arquiteturas superescalares, podendo ver de forma prática a interferência dos *pipelines* na performance de processamento do computador. Para assistir, acesse o *link* ou código a seguir.

https://goo.gl/aCq3Pf

Tipos de *pipeline*

São diversos os tipos de *pipeline* que podem ser implementados pelo processador para executar tarefas. Todos eles têm o mesmo objetivo: proporcionar maior agilidade e eficiência, reduzindo a ociosidade do processador. A seguir, vamos conhecer os principais tipos de *pipeline* e discutir suas características.

Fique atento

Para que os *pipelines* possam ser executados pela CPU, os *softwares* precisam estar programados para serem executados de forma paralela, explorando toda a capacidade do computador. Um exemplo de tecnologia utilizada para a construção de programas a serem executados utilizando paralelismo é o *Message Passing Interface* (MPI), um padrão para comunicação de dados em computação paralela.

Os *pipelines* se dividem da seguinte forma:

- *Pipelines* classificados por tipos de processamento:
  - *Pipeline* de instruções — é o *pipeline* descrito na seção anterior; refere-se ao processamento de instruções pelo processador do computador, podendo ser qualquer instrução; por exemplo, copiar um arquivo.
  - *Pipelines* aritméticos — são utilizados para a execução de operações aritméticas complexas — como multiplicação inteira, operações em ponto flutuante e operações vetoriais —, que podem ser divididas em etapas de execução menores; por exemplo, no caso de algoritmos de ordenação de vetores.
- *Pipelines* de controle de fluxo de dados:
  - Assíncrono — nesse método, os *pipelines* se comunicam por meio de sinais de *handshaking*, que são utilizados para indicar a disponibi-lidade de dados do estágio atual para o próximo estágio (RDY) e para indicar a liberação dos dados do estágio atual para o estágio anterior (ACK).
  - Síncrono — nesse estágio, os*pipelines* são interconectados por registradores (*latches*), que armazenam os dados intermediários durante a transferência entre estágios do *pipeline*. Nesse tipo de *pipeline*, toda transferência é controlada por um sinal de relógio, e o estágio com operação mais lenta determina a taxa de operação do *pipeline*.
- *Pipeline* de funcionalidade:
  - Unifuncional e multifuncional — *pipelines* unifuncionais são dedicados à execução de uma única operação, e *pipelines* multifuncionais podem executar mais de uma operação.
- *Pipeline* de estrutura:

- Linear — é um *pipeline* em que a execução dos ciclos segue um padrão sequencial, sem alterações de percurso.
- Não linear — é um *pipeline* em que o fluxo de execução dos estágios do *pipeline* sofre alterações de rota com frequência, podendo, inclusive, estabelecer novos ciclos.

Exemplo

Para entender melhor o conceito de *pipeline*, vamos pensar sob a perspectiva do *software*. Alguns *software* necessitam de grande capacidade computacional para atingir o seu objetivo, sendo que, algumas vezes, precisar de um grande tempo de espera para chegar ao resultado final. Existem diversos *software* desse tipo nas áreas de biologia e medicina, por exemplo.

Um exemplo de utilização de paralelismo são os algoritmos de ordenação de vetores. Imagine um vetor de 1 milhão de posições — se o processador efetuar a ordenação desse vetor de forma sequencial, vai precisar de um tempo significativo. Entretanto, se o programa quebrar o vetor em partes, e cada parte for executada separadamente por uma parte do processador, o tempo de ordenação do vetor será reduzido. Isso ocorre mesmo considerando-se que serão necessárias trocas de mensagens entre os estágios do *pipeline*, para garantir a ordenação total.

Referências

BRITO, A. V. *Introdução a arquitetura de computadores:* aumentando o desempenho com pipeline. [2018]. Disponível em: <http://producao.virtual.ufpb.br/books/edusantana/introducao-a-arquitetura-de-computadores-livro/livro/livro.chunked/ch02s07.html>. Acesso em: 18 dez. 2018.

CONHECENDO o Blog Opcode. [2018]. Disponível em: <http://opcode.com.br/opcode-blog/>. Acesso em: 18 dez. 2018.

MANACERO JR., A. *UCP e pipelines*. [2018]. Disponível em: <https://www.dcce.ibilce.unesp.br/~aleardo/cursos/arqcomp/pipelines.pdf>. Acesso em: 18 dez. 2018.

TANENBAUN, A. S. *Organização estruturada de computadores*. 5. ed. São Paulo: Pearson, 2007.

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.

Conteúdo: