# Organização de Computadores

**Prof. Robson de Souza**

## Nível de Microarquitetura

O projeto do nível de microarquitetura depende da ISA que está sendo implementada, bem como das metas de custo e desempenho do computador. Muitas ISAs modernas, em particular projetos RISC, têm instruções simples que normalmente podem ser executadas em um único ciclo de clock. ISAs mais complexas, como a Core i7, podem exigir muitos ciclos para executar uma única instrução. Executar uma instrução pode requerer localizar os operandos na memória, ler esses operandos e armazenar resultados de volta na memória.

### Exemplo de microarquitetura

O ideal seria que este tópico fosse apresentado explicando os princípios gerais do projeto de microarquitetura. infelizmente, não há princípios gerais; cada microarquitetura é um caso especial. Devido a isso, utilizaremos um exemplo específico para melhor entendimento.

Para nossa ISA que servirá de exemplo, escolhemos um subconjunto da Java Virtual Machine. Esse subconjunto contém somente instruções com inteiros, portanto, nós o denominamos IJVM para enfatizar que ele trata somente de inteiros.

Um modelo conveniente para o projeto de microarquitetura é pensar no projeto como um problema de programação no qual cada instrução no nível ISA é uma função a ser chamada por um programa mestre. Nesse modelo, o programa mestre é um laço simples, sem fim, que determina uma função a ser invocada, chama a função e então começa de novo.

O microprograma tem um conjunto de variáveis denominado **estado** do computador, que pode ser acessado por todas as funções. Cada função altera ao menos algumas das variáveis que compõem o estado. Por exemplo, o contador de programa (Program Counter – PC) é parte do estado. Ele indica a localização da memória que contém a próxima função (isto é, instrução ISA) a ser executada. Durante a execução de cada instrução, o PC é incrementado para indicar a próxima instrução a ser executada.

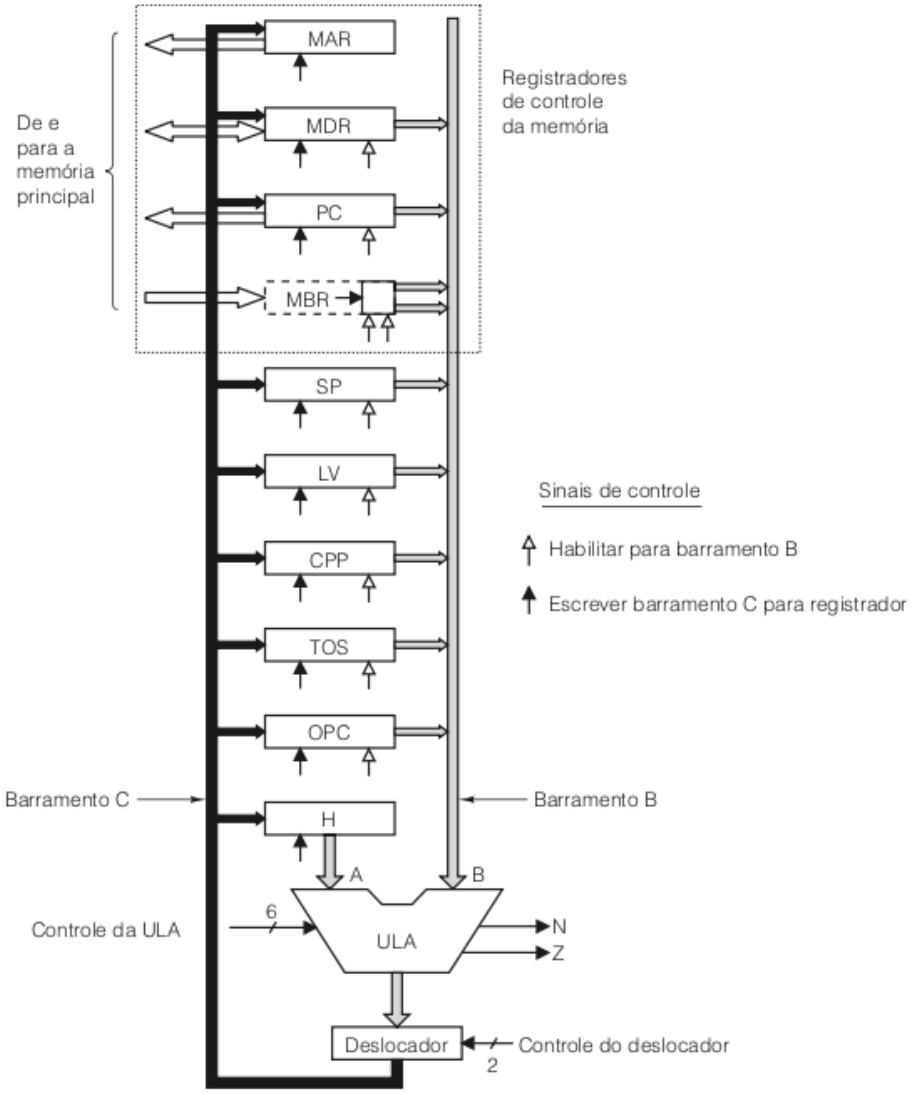
Instruções IJVM são curtas e fáceis. Cada instrução tem alguns campos, em geral um ou dois, e cada um deles tem alguma finalidade específica. O primeiro campo de toda instrução é o **opcode** (abreviatura de **operation code – código de operação**), que identifica a instrução, informando se ela é um ADD ou um BRANCH, ou qualquer outra coisa. Muitas instruções têm um campo adicional que especifica o operando. Por exemplo, instruções que acessam uma variável local precisam de um campo que identifique qual variável.

### O caminho de dados

O caminho de dados é a parte da CPU que contém a ULA, suas entradas e suas saídas. O caminho de dados de nossa microarquitetura de exemplo guarda uma razoável semelhança com o caminho de dados usado na maioria das máquinas. Contém vários registradores de 32 bits, aos quais atribuímos nomes simbólicos como PC, SP e MDR. Embora alguns desses nomes sejam familiares, é importante entender que esses registradores **são acessíveis apenas no nível de microarquitetura (pelo microprograma).**

A ULA é idêntica à que já vimos. Sua função é determinada por linhas de controle. No nosso exemplo, a ULA precisa de duas entradas de dados: uma entrada esquerda e uma entrada direita. Ligado à entrada esquerda está um registrador de retenção, H (pode ser carregado com a escolha de uma função da ULA). Ligado à entrada direita está um barramento, que pode ser carregado por cada uma das fontes.

A figura abaixo mostra o esquema do caminho de dados na nossa microarquitetura de exemplo.




Fonte: (Tanenbaum e Austin, 2013)

**Microinstruções**

Para controlar o caminho de dados, é necessário um conjunto de sinais, que podem ser divididos em cinco grupos funcionais:

1. Sinais para controlar escrita de dados do barramento C para registradores.
2. Sinais para controlar habilitação de registradores dirigidos ao barramento B para a entrada da ULA.
3. Sinais para controlar as funções da ULA e do deslocador.
4. Sinais (não mostrados) para indicar leitura/escrita na memória via MAR/MDR.
5. Sinal (não mostrado) para indicar busca na memória via PC/MBR.

Os valores dos sinais de controle especificam as operações para um ciclo do caminho de dados. Um ciclo consiste em copiar valores dos registradores para o barramento B, propagar os sinais pela ULA e pelo deslocador, dirigi-los ao barramento C e, por fim, escrever os resultados no registrador ou registradores adequados.

A sigla **MAR** se refere a **Memory Address Register (Registrador de Endereço da Memória)** e contém o endereço da posição da memória a ser lida ou escrita. Já a sigla **MDR** se refere a **Memory Data Register (Registrador de Dados da Memória)** e contém o dado a ser lido ou escrito na memória.

A sigla PC se refere ao Contador de Programa e a **MBR** se refere a **Memory Buffer Register (Registrador de Buffer da Memória)** e basicamente armazena os dados que estão sendo transferidos de e para a memória.

**Pilhas**

Praticamente todas as linguagens de programação trabalham com o conceito de procedimentos (métodos), que têm variáveis locais. Essas variáveis podem ser acessadas de dentro dos procedimentos, mas deixam de ser acessíveis assim que o procedimento é devolvido. Portanto, surge a pergunta: “Em que lugar da memória essas variáveis devem ser mantidas?”.

A solução mais simples, dar a cada variável um endereço de memória absoluto, não funciona. O problema é que um procedimento pode chamar a si mesmo (como um procedimento recursivo), desse modo, se um procedimento for ativado – isto é, chamado – duas vezes, é impossível armazenar suas variáveis em localizações absolutas de memória porque a segunda chamada irá interferir com a primeira.

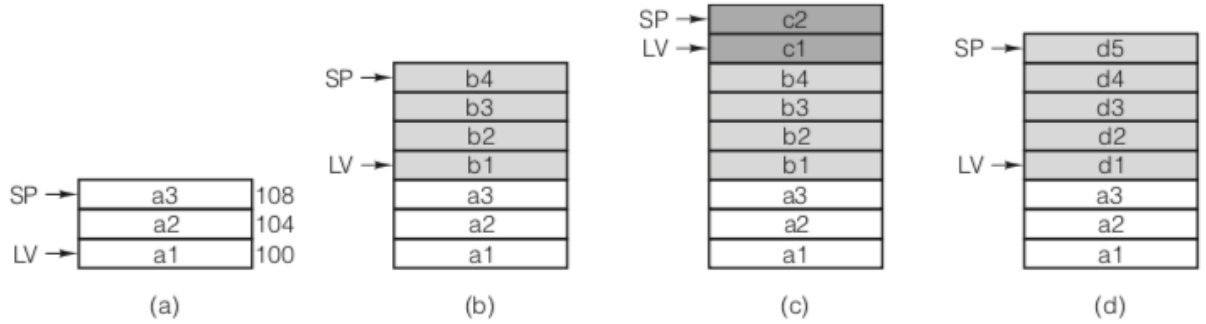
Em vez disso, é usada uma estratégia diferente. Uma área da memória, denominada pilha, é reservada para variáveis, mas variáveis individuais não obtêm endereços absolutos nela. Em vez disso, um registrador, por exemplo, LV, é preparado para apontar para a base das variáveis locais para o procedimento em questão.

Outro registrador, SP, aponta para a palavra mais alta das variáveis locais de A. Por exemplo, se LV for 100 e existirem 3 palavras com 4 bytes cada, então SP será 108. variáveis são referenciadas dando seu deslocamento (distância) em relação a LV. A estrutura de dados entre LV e SP (e incluindo ambas as palavras apontadas) é denominada quadro de variáveis locais de A.

Agora, vamos considerar o que acontece se A chamar outro procedimento, B. Onde deveriam ser armazenadas as quatro variáveis locais de B (b1, b2, b3, b4)? Resposta: na pilha, em cima das variáveis de A. Observe que LV deve ser ajustado pela chamada de procedimento para que aponte para as variáveis locais de B em vez das de A. As variáveis locais de B podem ser referenciadas dando seu deslocamento em relação a LV. De modo semelhante, se B chamar C, LV e SP são ajustados novamente para alocar espaço para as duas variáveis de C.

Quando C retorna, B torna-se ativo de novo e a pilha volta a ser ajustada, de modo que LV agora aponta outra vez para as variáveis locais de B. Da mesma forma, quando B retorna, voltamos à situação em que só existem as variáveis de A na pilha. Sob todas as condições, LV aponta para a base do quadro da pilha para o procedimento ativo no momento em questão e SP aponta para o topo do quadro da pilha.

Por exemplo, suponha que B e C retornaram e agora A chama D, que tem cinco variáveis locais. Essa é a situação na qual as variáveis locais de D usam a mesma memória que as de B usaram, bem como parte das de C. Com essa organização, a memória só é alocada para procedimentos que estão ativos no momento em questão. Quando um procedimento retorna, a memória usada por suas variáveis locais é liberada. A figura abaixo mostra os esquemas de pilha de A, B, C e D.

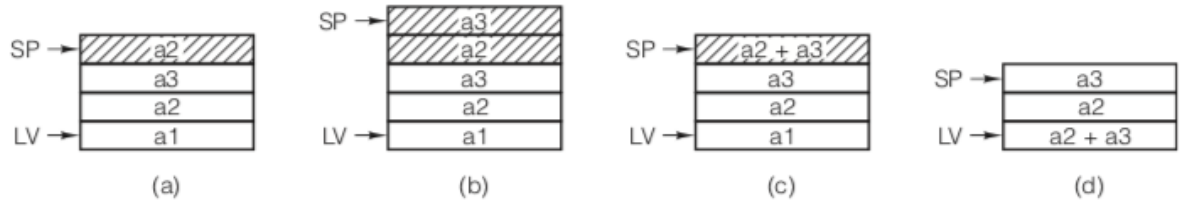



Fonte: (Tanenbaum e Austin, 2013)

Pilhas têm outra utilização além de conter variáveis locais. Elas podem ser usadas para reter operandos durante o cálculo de uma expressão aritmética. Quando usada dessa maneira, a pilha é denominada pilha de operandos. Suponha, por exemplo, que, antes de chamar B, A tenha de calcular a1 = a2 + a3;

Um modo de efetuar essa soma é passar a2 para a pilha, como ilustra a figura abaixo (a). Nesse caso, SP foi incrementado pelo número de bytes em uma palavra, por exemplo, 4, e o primeiro operando foi armazenado no endereço agora apontado por SP. Em seguida, a3 é passada para a pilha, conforme mostra a figura abaixo (b).

Agora, o cálculo propriamente dito pode ser feito executando uma instrução que retira duas palavras da pilha, soma as duas e devolve o resultado para a pilha, conforme a figura abaixo (c). Por fim, a palavra que está no topo pode ser retirada da pilha e armazenada de novo na variável local a1, como ilustrado na figura abaixo (d).




Fonte: (Tanenbaum e Austin, 2013)

Vale a pena observar que, enquanto todas as máquinas usam uma pilha para armazenar variáveis locais, nem todas usam uma pilha de operandos como essa para efetuar aritmética.

**Referências bibliográficas:**

TANENBAUM, Andrew S. Organização Estruturada de Computadores, 2007, 5ª Edição.

TANENBAUM, Andrew S. AUSTIN, Todd; Organização Estruturada de Computadores, 2013, 6ª Edição.