DCC006: Organização de computadores I

Trabalho Prático #3

Professor: Daniel Fernandes Macedo e Omar Paranaiba Vilela Neto

Antes de começar seu trabalho, leia todas as instruções abaixo.

- O trabalho deve ser feito em grupos de até 3 (três alunos). Cópias de trabalho acarretarão em devida penalização às partes envolvidas.
- Entregas após o prazo não serão aceitas.
- Submeta apenas um arquivo **.zip** contendo as suas soluções e um arquivo .txt com o nome e matrícula dos integrantes do grupo.
- A correção será **automatizada**, de forma que a saída do programa deve **seguir o formato apresentado nesta especificação**. Sigam todos os formatos dos números, o alinhamento da impressão, e os argumentos do programa a ser desenvolvido.

Simulador de Memória Cache

Neste trabalho, o objetivo é desenvolver um programa que funcione como um simulador de memória cache. O simulador será responsável por gerenciar uma cache de apenas um nível, processando acessos à memória RAM, preenchendo as linhas da cache conforme necessário e contabilizando a quantidade de *HITS* e *MISSES* ocorridos durante a execução. O programa deverá verificar se um determinado bloco de memória já está presente na cache (*HIT*) ou se precisa ser carregado da memória principal (*MISS*). Além disso, o simulador determinará em qual linha da cache um bloco de memória será armazenado, quando necessário.

Para simplificar o escopo do trabalho, o simulador não tratará os dados reais armazenados na memória. Em outras palavras, apenas o gerenciamento e o posicionamento das linhas na memória cache serão simulados, ignorando o conteúdo dos blocos.

Objetivo do trabalho

O principal objetivo deste trabalho é proporcionar a prática dos conceitos de **mapeamento direto**, **associatividade por conjunto** e **associatividade completa** que foram abordados durante o curso. A prática desses conceitos será essencial para compreender as diferenças entre os métodos de mapeamento e como eles afetam a eficiência e o desempenho da memória cache.

Especificação do simulador

O simulador deverá ser desenvolvido em **Python**, podendo utilizar qualquer biblioteca adicional que os alunos considerem necessária. O programa será executado via linha de comando e será testado em um ambiente Linux. Por essa razão, recomenda-se fortemente que o programa seja testado ao menos uma vez em uma máquina Linux antes da submissão, para evitar possíveis diferenças de comportamento entre sistemas operacionais, que podem causar falhas inesperadas durante a avaliação (acreditem, isso pode ocorrer!).

O programa deverá ser nomeado `simulador.py`. Ele receberá como argumentos os seguintes parâmetros, apresentados na ordem especificada:

1. **O tamanho total da cache, em bytes**. Se tivermos uma cache de 4KB, por exemplo, iremos digitar 4096.
2. **O tamanho de cada linha, em bytes**. Uma linha de 1KB, por exemplo, seria entrada digitando-se 1024.
3. **O tamanho de cada grupo, em unidades**. Pensando em uma memória cache de 4KB, e páginas de 1KB, teremos 4 linhas. Se tivermos uma linha por grupo, teremos um sistema de mapeamento direto. Se tivermos 4 por grupo, teremos um sistema de associatividade completa.

4. **O nome do arquivo** com os acessos à memória. O formato e o conteúdo do que terá nesse arquivo está especificado na seção "**O arquivo de entrada**" abaixo.

Parâmetros fixos do simulador

Além dos parâmetros fornecidos pela linha de comando, o simulador deve considerar os seguintes aspectos como fixos durante sua execução:

1. O espaço de endereçamento será de **32 bits**;
2. Os endereços sempre referenciam **bytes**, e não palavras;
3. A política de substituição de páginas é a **FIFO (*First In First Out*);**
4. A alocação de linhas em um conjunto seguirá a ordem. Ou seja, se tivermos um conjunto de tamanho dois, vamos primeiro armazenar o bloco na primeira linha do conjunto, e em seguida na segunda.
5. As linhas de um conjunto serão armazenadas de **forma consecutiva** na cache (uma atrás da outra).
6. Teremos **menos de mil linhas** de cache.:

Execução do programa durante a correção automática

O arquivo principal do simulador deve ser chamado **`simulador.py`**. O simulador será executado diretamente a partir da linha de comando usando o interpretador `python3`. O formato do comando será o seguinte:

```
python3 simulador.py 4096 1024 1 arquivo_de_entrada.txt
```

Onde **`4096`** é o tamanho total da cache em bytes, **`1024`** é o tamanho de cada linha em bytes, **`1`** é o tamanho de cada grupo em unidades e o **`arquivo_de_entrada.txt`** é o caminho para o arquivo de entrada que será utilizado como teste. Note que esses valores são apenas demonstrativos e a correção utilizará valores de entrada diferentes para analisar a corretude da solução.

Especificação do arquivo de entrada

O simulador recebe um arquivo de entrada em formato texto, contendo uma sequência de endereços de memória a serem acessados pela cache. Cada linha do arquivo representa um único endereço de memória, no formato hexadecimal, precedido por `0x`. Os endereços devem ser expressos em letras maiúsculas e conter exatamente 8 dígitos após o prefixo, representando os 32 bits do espaço de endereçamento.

Segue abaixo um exemplo válido de arquivo de entrada:

```
0xDEADBEEF
0x00000000
0x12345678
0xDEADBEEF
```

No exemplo acima, os endereços indicam que o simulador deverá acessar os blocos de memória correspondentes aos endereços `0xDEADBEEF`, `0x00000000`, `0x12345678` e novamente `0xDEADBEEF`, nesta ordem. Esses endereços serão usados para determinar se os dados já estão presentes na cache (*HIT*) ou se será necessário carregar o bloco da memória principal (*MISS*).

Ao implementar a leitura do arquivo, recomenda-se que o programa leia cada linha como um endereço hexadecimal e o converta para um valor numérico interno para realizar os cálculos necessários. Este arquivo de entrada será fornecido durante os testes do simulador, e o programa deve ser capaz de processá-lo corretamente, independentemente da quantidade de endereços ou da ordem em que aparecem.

Especificação da saída do simulador

A saída do simulador desenvolvido deverá ser gerada em um arquivo de texto chamado **`output.txt`**. Este arquivo conterá o estado das linhas da cache a cada acesso de memória, além da contagem total de *HITS* e *MISSES* ao final. A

saída será formatada de acordo com o exemplo apresentado a seguir (para fins demonstrativos, exemplo mostrado considera parâmetros: cache de 4KB, com linhas de 1KB cada, e associatividade completa).

A cada acesso de memória, o simulador imprimirá o estado das linhas da cache em blocos formatados. Cada bloco inicia com uma linha separadora composta por símbolos `=`. As colunas da tabela incluem o índice da linha na cache, o bit de validade e o identificador do bloco armazenado na linha, se aplicável.

O **índice da linha** é representado como um número inteiro de três dígitos, começando em zero e indo até o último valor de linha. Este índice representa a posição física da linha dentro da cache. Em uma memória cache com associatividade maior que 1, os índices serão usados para agrupar as linhas em conjuntos. Por exemplo, para associatividade 2, as linhas 000 e 001 pertenceriam ao primeiro conjunto, enquanto as linhas 002 e 003 pertenceriam ao segundo conjunto.

O **bit de validade** indica se a linha contém dados válidos. Este valor será 1 para linhas preenchidas com um bloco válido de memória e 0 para linhas que ainda não contêm nenhum bloco válido. Quando o bit de validade for 0, o endereço do bloco não deverá ser exibido.

O **identificador do bloco** é um número em hexadecimal de 32 bits, exibido sempre com caracteres em maiúsculas. Mesmo que o identificador de bloco real possua menos bits, ele será exibido no formato completo de 32 bits por simplicidade de implementação. O identificador do bloco é calculado a partir do endereço fornecido, eliminando os bits que representam o deslocamento dentro de uma linha da cache e os bits utilizados para identificar o conjunto (caso a cache tenha associatividade maior que 1).

O endereço da memória principal é fornecido como um valor hexadecimal de 32 bits. Ele representa uma posição na memória, mas inclui informações que não são relevantes para identificar o bloco na cache. Por exemplo: `0xDEADBEEF, 0x00000000, 0x12345678.`

As linhas da cache possuem um tamanho fixo, e os bits menos significativos do endereço correspondem à posição dentro da linha (deslocamento). Para calcular o identificador do bloco, esses bits devem ser eliminados. Se cada linha da cache tem 1KB (1024 bytes), são necessários 10 bits para endereçar qualquer posição dentro da linha. Portanto, os 10 bits menos significativos devem ser ignorados.

Exemplo:

Para `0xDEADBEEF`:
Binário: `11011110101011011011111011101111`
Ignorando os 10 bits menos significativos: `1101111010101101101111`**`0000000000`** → `0xDEADBC00`

Para `0x12345678`:
Binário: `00010010000110100010101011001111000`
Ignorando os 10 bits menos significativos: `0001001000011010001010`**`1000000000`** → `0x12345400`

Para `0x00000000`:
Já possui os 10 bits menos significativos como 0, então permanece igual: `0x00000000`.

Se a cache tiver associatividade maior que 1, os bits usados para identificar o conjunto também serão ignorados. Para uma cache de associatividade total, como no exemplo fornecido, não há conjuntos, então nenhum bit adicional precisa ser descartado. Se fosse uma cache com associatividade, o número de conjuntos determinaria quantos bits adicionais seriam ignorados. Por exemplo:

- Em uma cache com 4 conjuntos, seriam necessários 2 bits para identificar os conjuntos. Esses bits também seriam ignorados junto com os 10 bits de deslocamento.

Após ignorar os bits de deslocamento e os bits do conjunto (se aplicável), o valor restante representa o identificador do bloco. Para simplificar, ele será formatado como um número hexadecimal de 32 bits, com os caracteres em maiúsculas, independentemente do número real de bits.

Exemplos finais:

`0xDEADBC00` → identificador: `0x0037AB6F`

`0x00000000` → identificador: `0x00000000`

`0x12345400` → identificador: `0x00048D15`

Após a tabela que descreve o estado da cache, duas linhas adicionais indicam o número total de *HITS* e *MISSES* observados até aquele momento. Estas métricas são acumuladas ao longo dos acessos e devem ser exibidas no formato `#hits: X` e `#miss: Y`, onde `X` e `Y` são os valores acumulados.

Abaixo está um exemplo completo de saída para uma cache de 4KB, com linhas de 1KB, associatividade completa e os endereços de entrada `0xDEADBEEF`, `0x00000000`, `0x12345678` e `0xDEADBEEF`:

```
================
IDX V ** ADDR **
000 1 0x0037AB6F
001 0
002 0
003 0
================
IDX V ** ADDR **
000 1 0x0037AB6F
001 1 0x00000000
002 0
003 0
================
IDX V ** ADDR **
000 1 0x0037AB6F
001 1 0x00000000
002 1 0x00048D15
003 0
================
IDX V ** ADDR **
000 1 0x0037AB6F
001 1 0x00000000
002 1 0x00048D15
003 0

#hits: 1
#miss: 3
```

Documentação

Não estamos esperando nenhuma documentação escrita explicando o código, como o simulador funciona, etc.... A única coisa que deve ser entregue na documentação é um arquivo texto com o nome e matrícula dos alunos e, caso necessário, a lista de comandos que devem ser executados para executar o seu programa.

Dicas e sugestões finais

- Não deixe o trabalho para o último dia. Procurem os monitores e coloquem suas dúvidas no fórum do moodle. Não viva perigosamente!
- Confira o formato da saída do seu programa. A correção automática não perdoa!

Outros exemplos

Exemplo 1

Parâmetros: **`4096 1024 4`**

Arquivo de entrada	Saída esperada
0x0CB886CA 0x06BC89BA 0x379CBAD1 0x28F3123B 0x06F98ED5	================ IDX V ** ADDR ** 000 1 0x00032E21 001 0 002 0 003 0 ================ IDX V ** ADDR ** 000 1 0x00032E21 001 1 0x0001AF22 002 0 003 0 ================ IDX V ** ADDR ** 000 1 0x00032E21 001 1 0x0001AF22 002 1 0x000DE72E 003 0 ================ IDX V ** ADDR ** 000 1 0x00032E21 001 1 0x0001AF22 002 1 0x000DE72E 003 1 0x000A3CC4 ================ IDX V ** ADDR ** 000 1 0x0001BE63 001 1 0x0001AF22 002 1 0x000DE72E 003 1 0x000A3CC4 #hits: 0 #miss: 5

Exemplo 2

Parâmetros: **`4096 1024 4`**

Arquivo de entrada	Saída esperada
0x0CB886CA 0x06BC89BA 0x379CBAD1 0x28F3123B 0x06F98ED5 0x06F98ED8 0x06BC89BA	================ IDX V ** ADDR ** 000 1 0x00032E21 001 0 002 0 003 0 ================ IDX V ** ADDR ** 000 1 0x00032E21 001 1 0x0001AF22 002 0 003 0 ================ IDX V ** ADDR ** 000 1 0x00032E21 001 1 0x0001AF22 002 1 0x000DE72E 003 0

	`` ================ IDX V ** ADDR ** 000 1 0x00032E21 001 1 0x0001AF22 002 1 0x000DE72E 003 1 0x000A3CC4 ================ IDX V ** ADDR ** 000 1 0x0001BE63 001 1 0x0001AF22 002 1 0x000DE72E 003 1 0x000A3CC4 ================ IDX V ** ADDR ** 000 1 0x0001BE63 001 1 0x0001AF22 002 1 0x000DE72E 003 1 0x000A3CC4 #hits: 2 #miss: 5 ```

Exemplo 3

Parâmetros: **`4096 1024 4`**

Arquivo de entrada	Saída esperada
`` 0x0CB886CA 0x06BC89BA 0x379CBAD1 0x28F3123B 0x06F98ED5 0x06F98ED8 0x17AA08A2 ```	``` ================ IDX V ** ADDR ** 000 1 0x00032E21 001 0 002 0 003 0 ================ IDX V ** ADDR ** 000 1 0x00032E21 001 1 0x0001AF22 002 0 003 0 ================ IDX V ** ADDR ** 000 1 0x00032E21 001 1 0x0001AF22 002 1 0x000DE72E 003 0 ================ IDX V ** ADDR ** 000 1 0x00032E21 001 1 0x0001AF22 002 1 0x000DE72E 003 1 0x000A3CC4 ================ IDX V ** ADDR ** 000 1 0x0001BE63 001 1 0x0001AF22 002 1 0x000DE72E 003 1 0x000A3CC4 ================ IDX V ** ADDR ** 000 1 0x0001BE63 001 1 0x0001AF22 002 1 0x000DE72E 003 1 0x000A3CC4 ================ IDX V ** ADDR ** 000 1 0x0001BE63 001 1 0x0005EA82 002 1 0x000DE72E 003 1 0x000A3CC4 ```

	#hits: 1 #miss: 6

Exemplo 4

Parâmetros: **`4096 1024 4`**

Arquivo de entrada	Saída esperada
0x0CB886CA 0x06BC89BA 0x379CBAD1 0x28F3123B 0x06F98ED5 0x06F98ED8 0x17AA08A2 0x06BC89BA	========== IDX V ** ADDR ** 000 1 0x00032E21 001 0 002 0 003 0 ========== IDX V ** ADDR ** 000 1 0x00032E21 001 1 0x0001AF22 002 0 003 0 ========== IDX V ** ADDR ** 000 1 0x00032E21 001 1 0x0001AF22 002 1 0x000DE72E 003 0 ========== IDX V ** ADDR ** 000 1 0x00032E21 001 1 0x0001AF22 002 1 0x000DE72E 003 1 0x000A3CC4 ========== IDX V ** ADDR ** 000 1 0x0001BE63 001 1 0x0001AF22 002 1 0x000DE72E 003 1 0x000A3CC4 ========== IDX V ** ADDR ** 000 1 0x0001BE63 001 1 0x0001AF22 002 1 0x000DE72E 003 1 0x000A3CC4 ========== IDX V ** ADDR ** 000 1 0x0001BE63 001 1 0x0005EA82 002 1 0x000DE72E 003 1 0x000A3CC4 ========== IDX V ** ADDR ** 000 1 0x0001BE63 001 1 0x0005EA82 002 1 0x0001AF22 003 1 0x000A3CC4 #hits: 1 #miss: 7

Exemplo 5

Parâmetros: **`4096 1024 2`**

Arquivo de entrada	Saída esperada
`0x0CB886CA` `0x06BC89BA`	`===============` `IDX V ** ADDR **` `000 1 0x00032E21` `001 0` `002 0` `003 0` `===============` `IDX V ** ADDR **` `000 1 0x00032E21` `001 0` `002 1 0x0001AF22` `003 0` `#hits: 0` `#miss: 2`

Exemplo 6

Parâmetros: **`4096 1024 1`**

Arquivo de entrada	Saída esperada
`0x0CB886CA` `0x06BC89BA` `0x379CBAD1` `0x28F3123B` `0x06F98ED5` `0x06F98ED8` `0x17AA08A2`	`===============` `IDX V ** ADDR **` `000 0` `001 1 0x00032E21` `002 0` `003 0` `===============` `IDX V ** ADDR **` `000 0` `001 1 0x00032E21` `002 1 0x0001AF22` `003 0` `===============` `IDX V ** ADDR **` `000 0` `001 1 0x00032E21` `002 1 0x000DE72E` `003 0` `===============` `IDX V ** ADDR **` `000 1 0x000A3CC4` `001 1 0x00032E21` `002 1 0x000DE72E` `003 0` `===============` `IDX V ** ADDR **` `000 1 0x000A3CC4` `001 1 0x00032E21` `002 1 0x000DE72E` `003 1 0x0001BE63` `===============` `IDX V ** ADDR **` `000 1 0x000A3CC4` `001 1 0x00032E21` `002 1 0x000DE72E` `003 1 0x0001BE63` `===============` `IDX V ** ADDR **` `000 1 0x000A3CC4` `001 1 0x00032E21` `002 1 0x0005EA82` `003 1 0x0001BE63`

	#hits: 1 #miss: 6