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 iremos desenvolver um programa simulador de memórias cache. Vamos fazer uma cache de somente um nível, que recebe acessos de memória e faz o preenchimento das suas linhas **e conte a quantidade de HITS e MISS**. **O programa desenvolvido irá verificar se um certo bloco de memória RAM já está preenchido nas linhas da memória cache (HITS) ou não (MISS), bem como determinar em qual linha um certo bloco de memória RAM será armazenado**. O nosso simulador é bem simples, de forma que iremos considerar somente o posicionamento das linhas na memória cache. Os dados em si serão ignorados, e desta forma não serão simulados.

O objetivo do trabalho é praticar os conceitos de associatividade completa, associatividade por conjunto e mapeamento direto. Vimos no curso que a associatividade por conjunto é a forma mais geral de todas: se temos somente um conjunto com todas as linhas, então a memória funciona como associatividade completa. Se temos conjuntos com somente uma linha de cache, então temos o sistema de mapeamento direto.

O trabalho será corrigido de forma automática, usando um programa com vários testes. Os testes começarão de forma básica, verificando a inserção de umas poucas linhas de cache, e irão evoluir para testes mais complexos onde a política de substituição de páginas será avaliada.

Especificação do Simulador

O simulador poderá ser desenvolvido em C, C++ ou Python, empregando qualquer biblioteca desejada pelos alunos. O programa deve ser executado em linha de comando, e será testado em uma máquina Linux. Recomenda-se que os alunos testem o programa pelo menos uma vez antes da submissão em uma máquina Linux. Isso evita eventuais diferenças de comportamento entre os sistemas operacionais (acreditem, isso pode ocorrer!). O programa, que após ser compilado deverá ter o nome **`simulador (ou simulador.py)`**, receberá os seguintes argumentos:

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.

Além dos dados de execução, considere os seguintes parâmetros como sendo **fixos**:

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 terceira.
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.

Dicas:

1. Caso usem C, uma forma prática de representar os valores de memória é usando o tipo **uint32_t**, que define um inteiro de 32 bits sem sinal. Esse tipo é definido na biblioteca **<sys/types.h>**.

O arquivo de entrada

O arquivo de entrada do simulador será um arquivo texto, contendo um endereço de memória a ser acessado. Os dados serão codificados em hexadecimal, como o abaixo:

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

Dicas:

1. Caso usem C (ou mesmo no python), o **`fscanf`** lê números em hexadecimal usando o **`"%x"`**. Esse parâmetro espera um inteiro sem sinal como variável que irá receber os valores.

A Saída do simulador

O simulador desenvolvido deverá obedecer o espaçamento e a formatação das saídas como apresentado abaixo. Iremos apresentar as linhas da cache, qual bloco da memória está em cada linha, e finalmente se aquela linha é válida ou não. Para a entrada apresentada na seção anterior, considerando uma cache de 4KB, com linhas de 1KB cada, e associatividade completa, teríamos a seguinte saída:

```
================
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

#hits: 1
#miss: 3
```

A saída deverá ser feito em um arquivo de texto **nomeado como output.txt**

Reparem que a saída consiste de um **índice**, começando em zero e chegando até o último valor de linha. Considere que o índice vai contar as linhas. Para ilustrar, uma memória cache com associatividade 2 iria ter dois conjuntos, sendo um com as linhas 000 e 001, e outro com as linhas 002 e 003.O índice deve ser impresso como um inteiro de 3 dígitos.

O segundo campo é o bit de validade da linha. Ele deve ser zero se a linha não tiver dados válidos, e um se tiver dados válidos. Quando tivermos dados inválidos, o endereço do bloco não deve ser impresso.

O terceiro campo será o identificador do bloco. Para uniformizar a entrada, este campo deve ser um inteiro, em hexadecimal, contendo 32 bits, com os caracteres em maiúsculas. Iremos apresentar 32 bits mesmo tendo menos bits para identificador de bloco, a título de simplificação da implementação. Lembrem-se que **não vamos armazenar os bits identificadores de bloco**, porque eles são implícitos pelo próprio número do conjunto. A título de exemplo, para a nossa execução iríamos eliminar os 10 bits menos significativos (porque eles seriam os deslocamentos dentro de uma página de 1KB), assim:

```
0xDEADBEEF vira 0xDEADBC00
0x00000000 vira 0x00000000
0x12345678 vira 0x12345400
```

Finalmente, iremos ignorar os dez bits menos significativos e os bits de identificação de bloco para obter o identificador de bloco.

```
0xDEADBC00 vira 0x0037AB6F
0x00000000 vira 0x00000000
0x12345400 vira 0x00048D15
```

As duas últimas linhas da saída deverão ser, respectivamente, a quantidade de hits e miss no formato exemplificado.

Dicas:

1. Caso usem C (ou mesmo no python), o **`printf`** tem uma forma muito bacana para imprimir um número de x dígitos. Por exemplo, `printf("%3d", indice)` imprime um número inteiro com 3 dígitos.

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 **a lista de comandos que devem ser executados para compilar o seu programa no Linux**, bem como uma linha mostrando como executar o seu programa. Este mesmo arquivo texto irá informar o nome e matrícula dos membros do grupo.

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!