Especificação do Trabalho – Implementação de um Simulador de Caches

Objetivos:

- Implementação de um simulador de caches parametrizável em uma linguagem de programação.
- Os grupos serão compostos por 2 discentes.

Implementação do Simulador

Deverá ser implementado um simulador funcional de caches em uma linguagem de programação (livre escolha), este simulador deverá ser parametrizável (configurações da cache), quanto ao número de conjuntos, tamanho do bloco, nível de associatividade e política de substituição. Considere que a cache é endereçada à bytes e o endereço possui 32 bits.

A configuração de cache deverá ser **repassada por linha de comando** e formatada com os seguintes parâmetros (o arquivo de entrada poderá ter extensão):

cache_simulator <nsets> <bsize> <assoc> <substituição> <flag_saida> *arquivo_de_entrada*

Onde cada um destes campos possui o seguinte significado:

- ***cache_simulator*** - nome do arquivo de execução principal do simulador (todos devem usar este nome, independente da linguagem escolhida;
- ***nsets*** - número de conjuntos na cache (número total de "linhas" ou "entradas" da cache);
- ***bsize*** - tamanho do bloco em bytes;
- ***assoc*** - grau de associatividade (número de vias ou blocos que cada conjunto possui);
- ***substituição*** - política de substituição, que pode ser Random (R), FIFO (F) ou L (LRU);
- ***flag_saida*** - flag que ativa o modo padrão de saída de dados;
- ***arquivo_de_entrada*** - arquivo com os endereços para acesso à cache.

O tamanho da cache é dado pelo produto do número de conjuntos na cache (<nsets>), tamanho do bloco em bytes (<bsize>) e associatividade (<assoc>).

Para caches associativas, o simulador deverá oferecer ao usuário a possibilidade de escolha entre as políticas de substituição randômica, FIFO e LRU.

Formato de saída:

A saída do simulador será um relatório de estatísticas com o número total de acessos, número total *hits* e *misses* (os *misses* deverão ser classificados em compulsórios, capacidade e conflito), que poderá ser exibido em dois formatos, de acordo com o valor do campo *flag_saida*:

- *flag_saida* = 0
  - Formato livre, pode conter textos com labels para facilitar a visualização. Ex: Taxa de hits = 90%.
- *flag_saida* = 1
  - Formato padrão de saída que deverá respeitar a seguinte ordem: Total de acessos, Taxa de hit, Taxa de miss, Taxa de miss compulsório, Taxa de miss de capacidade, Taxa de miss de conflito
  - Ex: 100000, 0.95, 0.06, 0.17 0.33 0.50

Arquivo de entrada:

O arquivo de entrada será utilizado como entrada para o simulador (armazenado em formato binário) que conterá os endereços requisitados à cache (**cada endereço possui 32 bits, em formato *big endian***). Quatro arquivos de teste serão fornecidos para auxiliar na verificação dos códigos, contendo 100, 1000, 10.000 e 186676 endereços. Cada um deles estará disponível no formato binário exigido, e também em .txt para facilitar a visualização dos valores e o entendimento do comportamento da cache.

Exemplo:
Considerando a seguinte linha de comando, utilizando o arquivo de entrada "bin_100.bin":

cache_simulator 256 4 1 R 1 bin_100.bin

O resultado esperado para a saída é: 100 0.92 0.08 1.00 0.00 0.00

Envio dos códigos:

Cada dupla deverá compactar **APENAS** o código fonte (e executáveis, caso haja) em um arquivo, nomeado com o número de matrícula de um dos membros da dupla, e enviar o arquivo pelo e-aula.
Ex: 12345678.zip

Recomendo que um arquivo "readme.txt" seja incluído neste arquivo compactado, descrevendo as eventuais funcionalidades adicionais, bem como eventuais informações úteis para a compilação (precisa de alguma biblioteca específica?) e execução do código.

Avaliação:

A avaliação será realizada em duas etapas a partir dos códigos enviados. **A primeira etapa (peso 5**), será automatizada, identificando códigos "muito similares" com os demais entregues (neste semestre ou em semestres anteriores) e rodando testes preliminares com alguns benchmarks. Depois disso, serão executados testes padrão para todos os códigos e os resultados serão verificados com os valores esperados. **A segunda etapa (peso 5)** de avaliação será a apresentação do código ao professor, feita pelas duplas em sala de aula conforme cronograma divulgado no e-aula. Nesta **segunda etapa**, **as notas serão individuais**, ou seja, cada membro da dupla deverá responder corretamente as perguntas sobre o código desenvolvido.

Pseudocódigo de exemplo para começar o simulador

Linha de comando simplificada: *cache_simulator* <nsets> <bsize> <assoc> *arquivo_de_entrada.bin*

```
cache_val [n_sets * assoc];
cache_tag [n_sets * assoc];
//criar uma estrutura de dados para armazenar os tags e os bits de validade.

n_bits_offset = log2 bsize;
n_bits_indice = log2 nsets;

n_bits_tag = 32 - n_bits_offset - n_bits_indice;
//descobre o número de bits de cada parcela do endereço

//para todos os endereços do arquivo
while (not EOF)
{
        endereço = ler (arquivo_de_entrada.bin);
        tag = endereço >> (n_bits_offset + n_bits_indice);
        indice = (endereço >> n_bits_offset) & (2^n_bits_indice -1);
            //isso é uma máscara que vai deixar apenas os bits do índice na variável "endereço".

        // para o mapeamento direto
        if (cache_val[indice] == 0)
        {
                miss_conpulsório ++;
                cache_val[indice] = 1;
                cache_tag[indice] = tag;
                // estas duas últimas instruções representam o tratamento da falta.
        }
        else
                if (cache_tag[indice] == tag)
                        hit ++;
                else
                {
                        miss ++;
                            //conflito ou capacidade?
                        cache_val[indice] = 1;
                        cache_tag[indice] = tag;
                }
        .
        .
        .
```