CENTRO TECNOLÓGICO
Universidade Federal de Santa Catarina

Laboratório 06

Simulador de Cache

Alunas: Jéssica Regina dos Santos e Myllena Corrêa
Disciplina: Organização de Computadores I
Professor: Marcelo Daniel Berejuck
Entrega: 28/05/25

Questão 01)

Como solicitado pelo professor, utilizamos o código da primeira questão do laboratório anterior (Laboratório 05). Foram realizados os passos descritos no enunciado deste laboratório (Laboratório 06).

```
.data # Declaracao de variaveis

# Matriz 16x16 a ser preenchida
data: .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

.text

main:
        la $s0, data
        li $t0, 0 # $t0 = value (valor a ser armazenado em data e incrementado em loop)
        li $t1, 0 # $t1 sera nosso contador de linhas (i)
        row_loop: # laco que incrementa i
                beq $t1, 16, exit_row # continua ate que i = 16
                li $t2, 0 # $t2 sera nosso contador de colunas (j)
                column_loop: # laco que incrementa j
                        beq $t2, 16, exit_column # continua ate que j = 16
                        mul $t3, $t1, 16 # $t3 = i x 16
                        add $t3, $t3, $t2 # $t3 = i x 16 + j
                        sll $t3, $t3, 2 # $t3 = (i x 16 + j) x 4
                        add $t3, $t3, $s0 # $t3 = (i x 16 + j) x 4 + endereco base de data = endereco de data[i][j]
                        sw $t0, 0($t3) # data[i][j] = value
                        addi $t0, $t0, 1 # value++
                        addi $t2, $t2, 1 # j++
                        j column_loop # reinicia laco
                exit_column: # fim do laco j
                addi $t1, $t1, 1 # i++
                j row_loop # reinicia laco
        exit_row: # fim do laco i

# programa finalizado
```

Imagem 1. Código da questão 1 (matriz que percorre linha a linha).

Pergunta 1) Qual foi a taxa final de acertos do cache?
Pergunta 2) Qual será a taxa de acertos se o tamanho do bloco for aumentado de 4 para 8 words?
Pergunta 3) E se for diminuído de 4 words para 2 words?

Respostas: A taxa de acerto na cache foi de 75% (Imagem 2).

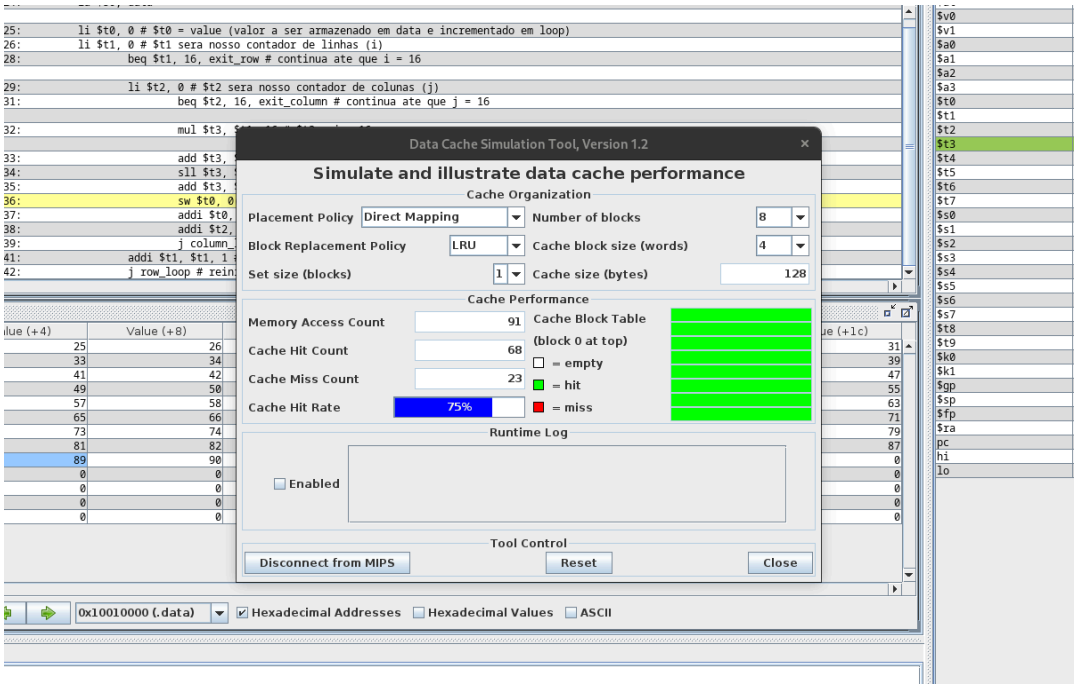

Imagem 2. Data Cache Performance para a primeira matriz.

Aumentando a cache para 8 blocos de 8 words, é visível um aumento de 75% para 88% de acertos (Imagem 3). Devido a localidade temporal, quando uma nova palavra é acessada de um novo bloco de memória, ocorre um miss. O bloco de 8 palavras é carregado para a cache e as próximas 7 consultas resultam em hits. Resumidamente, a cada 8 palavras, temos 1 miss e 7 hits. Na cache simulada anteriormente, isso não ocorria, pois o tamanho da cache era menor (de 4 words), resultando em, a cada 4 palavras carregadas, 3 hits.

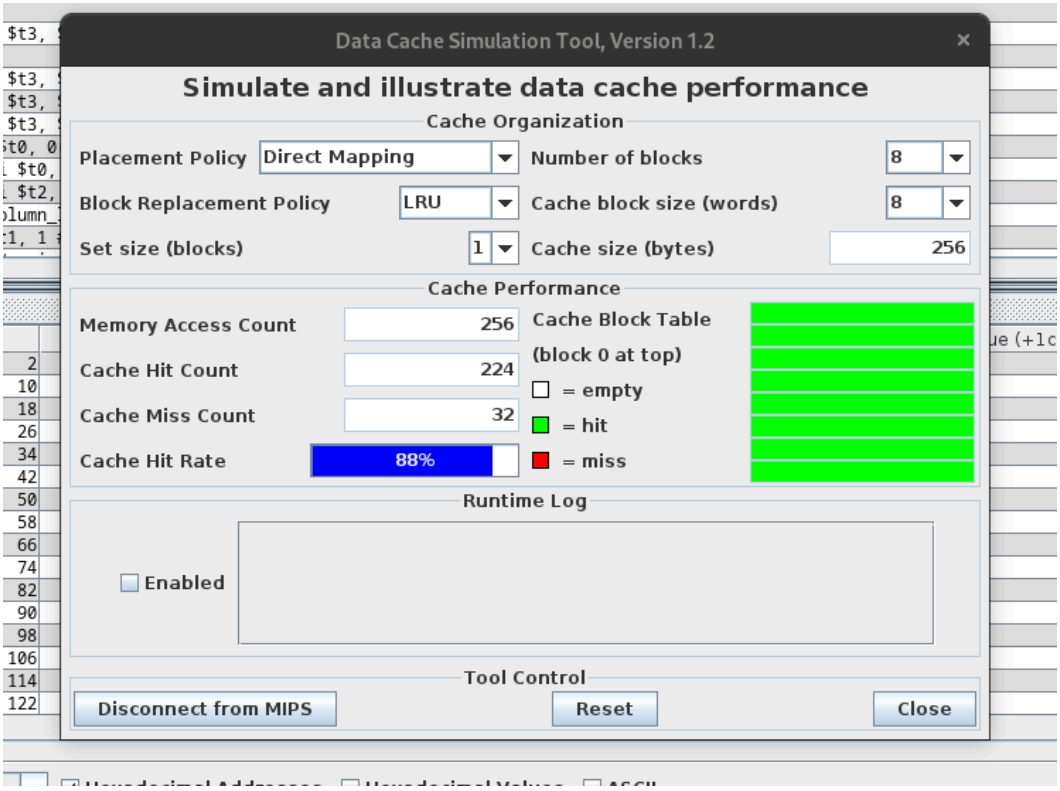

Imagem 3. 8x8.

Em seguida, foi aplicado uma nova configuração na cache: mudamos de 4 words por bloco para 2 words por bloco, como apresentado na Imagem 4, abaixo:

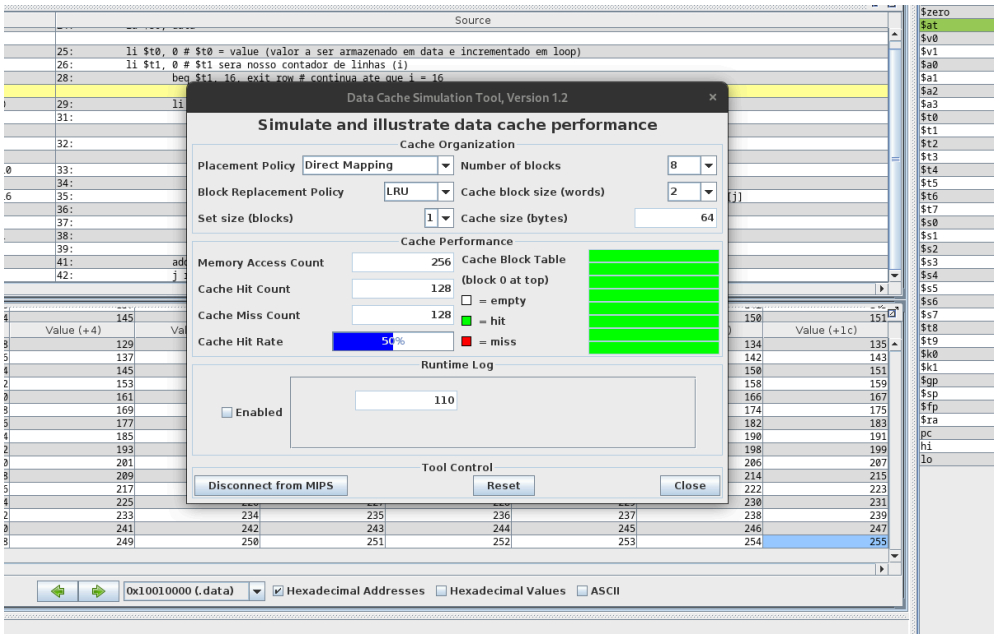

Imagem 4. Data Cache Performance para a primeira matriz, agora com duas words por bloco.

Realizando a mudança, é notável a diminuição de desempenho, sendo a taxa de acerto diminuída para 50%. Isso ocorreu pois foi aplicado uma configuração que diminuiu a

capacidade geral da cache; ou seja, o processador precisa buscar mais vezes os dados, que resultam em miss na memória.

Questão 2)

Pergunta 1) Qual foi o desempenho do cache para este programa (8 blocos, 4 words por bloco)?
Pergunta 2) [Após seguir as instruções no enunciado da questão] Qual é o desempenho do cache da instância da ferramenta original (tamanho de bloco e número de blocos igual a 16)?
Pergunta 3) Qual é o desempenho do cache da segunda instância da ferramenta?

Respostas:

Realizamos o mesmo procedimento para o Exercício 2 do laboratório anterior (Imagem 5), tendo uma cache de 8 blocos com 4 words cada. Encontramos como resultado 100% de miss rate (Imagem 6).

Nosso código acessa a matriz coluna por coluna e cada valor é guardado linha por linha na memória, fazendo com que cada acesso salte de uma região de 16 words, que o tamanho do bloco não consegue carregar. Praticamente, cada acesso precisa de um novo bloco, resultando em 100% de miss.

```
.data # Declaraco de variaveis

# Matriz 16x16 a ser preenchida
data: .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

.text

main:
        la $s0, data
        li $t0, 0
        li $t2, 0 # $t2 sera nosso contador de colunas (j)
        column_loop: # laco que incrementa j
                beq $t2, 16, exit_column # continua ate que j = 16
                li $t1, 0 # $t1 sera nosso contador de linhas (i)
                row_loop: # laco que incrementa i
                        beq $t1, 16, exit_row # continua ate que i = 16
                        mul $t3, $t1, 16 # $t3 = i x 16
                        add $t3, $t3, $t2 # $t3 = i x 16 + j
                        sll $t3, $t3, 2 # $t3 = (i x 16 + j) x 4
                        add $t3, $t3, $s0 # $t3 = (i x 16 + j) x 4 + endereco base de data = endereco de data[i][j]
                        sw $t0, 0($t3) # data[i][j] = value
                        addi $t0, $t0, 1 # value++
                        addi $t1, $t1, 1 # i++
                        j row_loop # reinicia laco
                exit_row: # fim do laco i
                addi $t2, $t2, 1 # j++
                j column_loop # reinicia laco
        exit_column: # fim do laco j

# programa finalizado
```

Imagem 5. Código da questão 2 (matriz que percorre coluna por coluna).

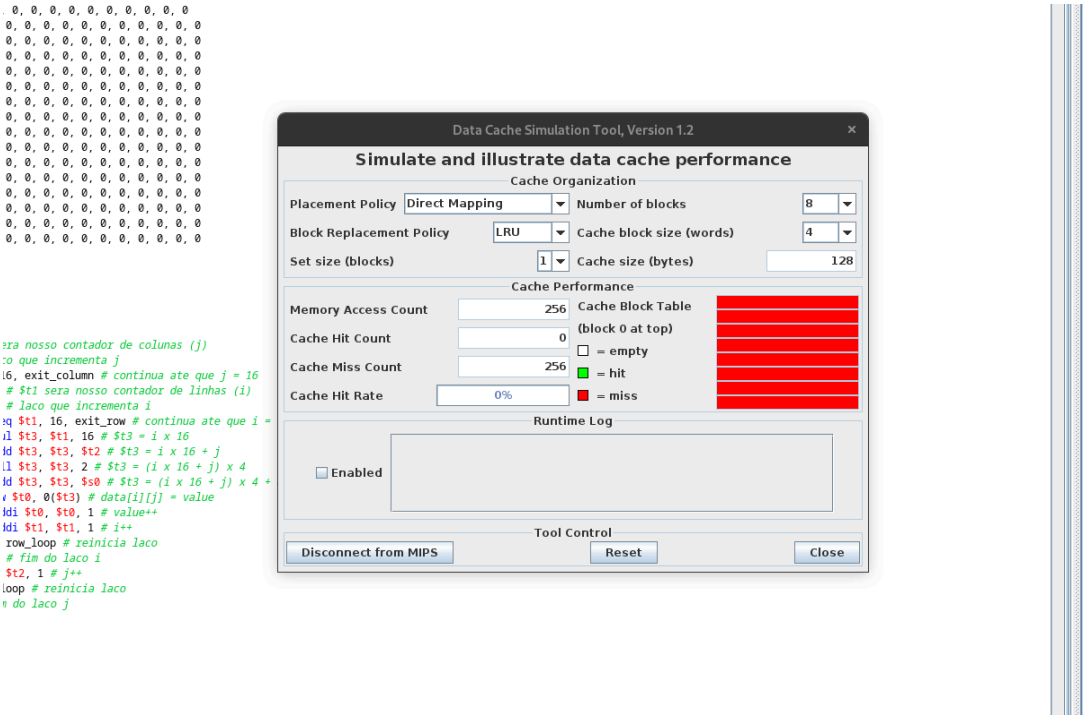

Imagem 6. Cache Hit Rate igual a 0%. Logo, 100% miss.

Em seguida, modificamos a primeira cache para 16 blocos. Também será simulado uma outra cache de 16 blocos com 16 words por bloco (Imagem 7).

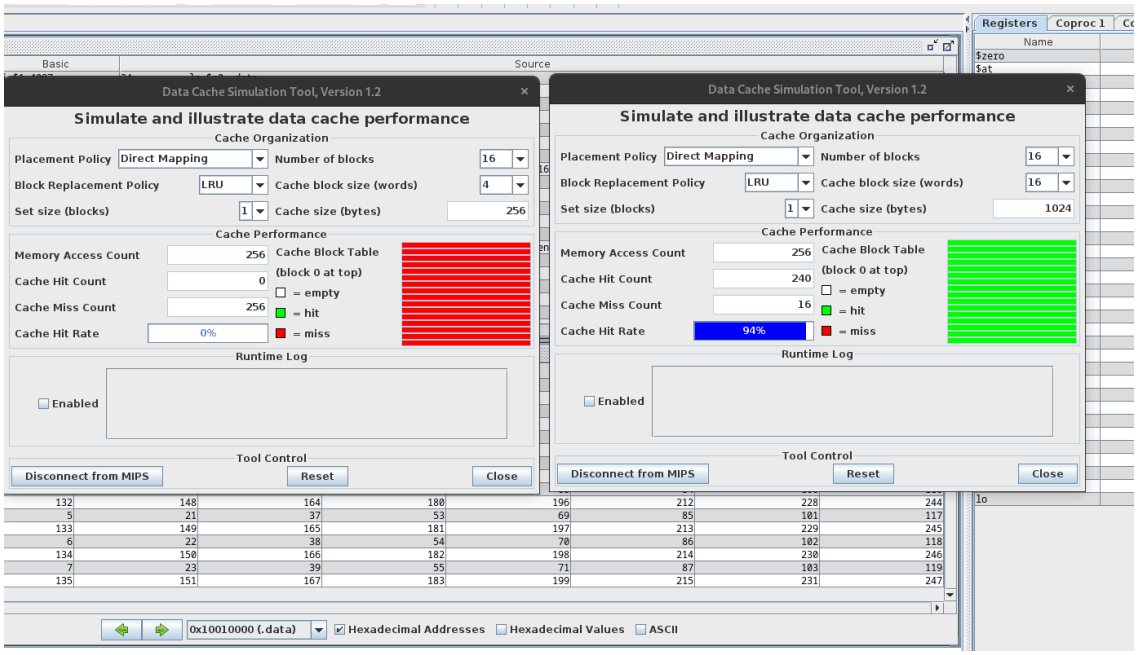

Imagem 7. Caches 16x4 e 16x16.

Concluímos que, mantendo os tamanhos de blocos de cache pequenos e aumentando a quantidade de blocos, ainda não mudaria nada, já que o tamanho do bloco ainda será insuficiente para conseguir carregar o bloco de 16 words para a cache.

Aumentando a quantidade de blocos e o tamanho do bloco, ocorre um resultado mais positivo. Ao todo ocorrem 16 miss, porém, em cada miss, será carregada uma linha inteira para a cache. Como a cache agora contém a matriz toda, os acessos às colunas seguintes encontram as linhas já carregadas, resultando em vários hits, totalizando 94% de acertos (hits).

Questão 3)

Concluímos que o experimento da **Questão 1** foi o mais eficiente, pois após a realização dos experimentos citados neste relatório e avaliações na taxa de falhas de cache (Cache Miss Rate) para algumas configurações (8x2, 8x4, 8x8, 16x4, 16x16, etc), vimos que o primeiro programa utiliza a cache de maneira mais eficiente que o segundo, isso porque ele tem taxas de erro/faltas menores (apesar de que os programas se igualam em algumas configurações, como a 16 blocos de 16 words). Portanto parece razoável dizer que o primeiro, muito provavelmente, tem melhor desempenho que o segundo.