Laboratório 5

INE5411 - Organização de Computadores I

Nome: Maykon Marcos Junior
Matrícula: 22102199

Exercício 1

A matriz A foi alocada com 1's e a matriz B com 2's, de forma que, para verificar o sucesso do código, os primeiros MAX² blocos devem ser 3, os próximos 2 e nenhum 1. O local de memória é reservado previamente, mas apenas como 0, sendo alocado depois

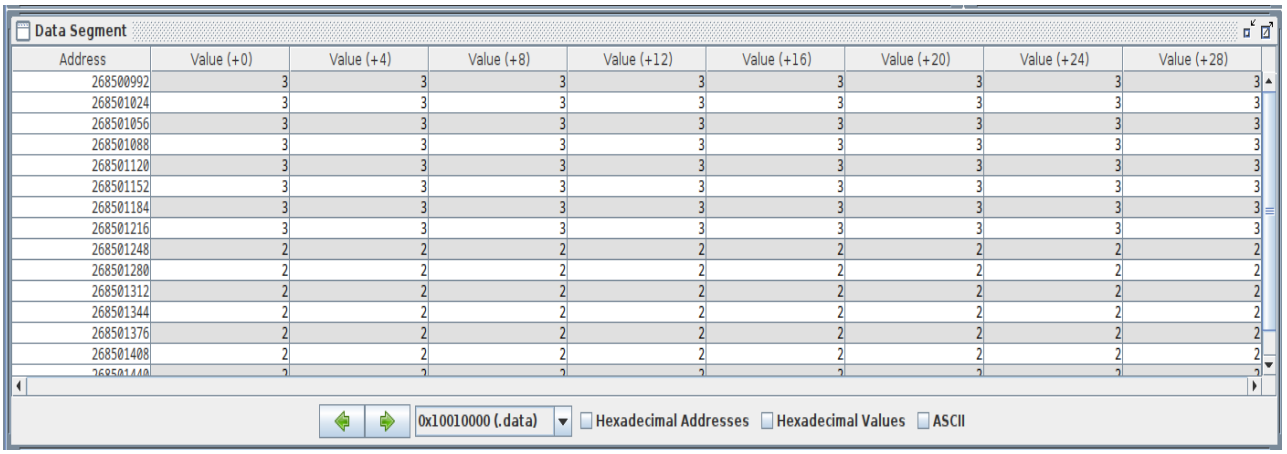
Teste com MAX = 4, primeiros 16 blocos = 3, e o resto = 2

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+12)	Value (+16)	Value (+20)	Value (+24)	Value (+28)
268500992	3	3	3	3	3	3	3	3
268501024	3	3	3	3	3	3	3	3
268501056	2	2	2	2	2	2	2	2
268501088	2	2	2	2	2	2	2	2
268501120	0	0	0	0	0	0	0	0
268501152	0	0	0	0	0	0	0	0
268501184	0	0	0	0	0	0	0	0
268501216	0	0	0	0	0	0	0	0
268501248	0	0	0	0	0	0	0	0
268501280	0	0	0	0	0	0	0	0
268501312	0	0	0	0	0	0	0	0
268501344	0	0	0	0	0	0	0	0
268501376	0	0	0	0	0	0	0	0
268501408	0	0	0	0	0	0	0	0

0x10010000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII

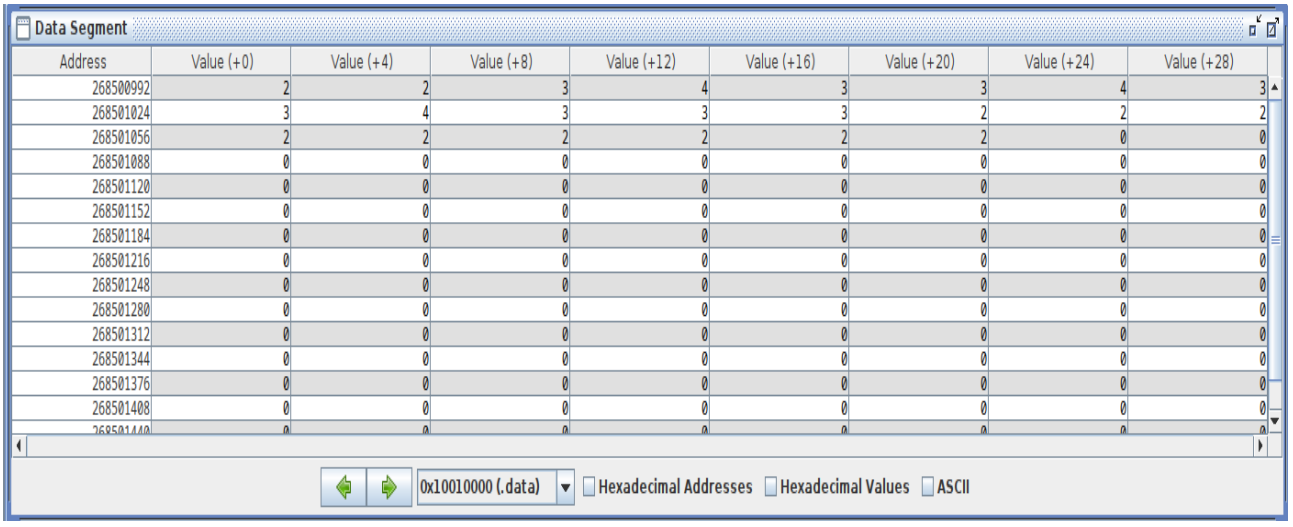
Teste com MAX = 8



Exercício 2

O código em linguagem de maior nível apresentado não checa se as variáveis ii e jj ultrapassam o valor MAX, além de não inicializar a matriz, então o código resultante não será equivalente ao primeiro, com várias atribuições repetidas e fora dos limites, especialmente quando MAX não é múltiplo de blocksize e, embora não seja muito comum nem faça muito sentido, se este for maior que aquele

Ex com MAX = 3 e blocksize = 2



Exercício 3

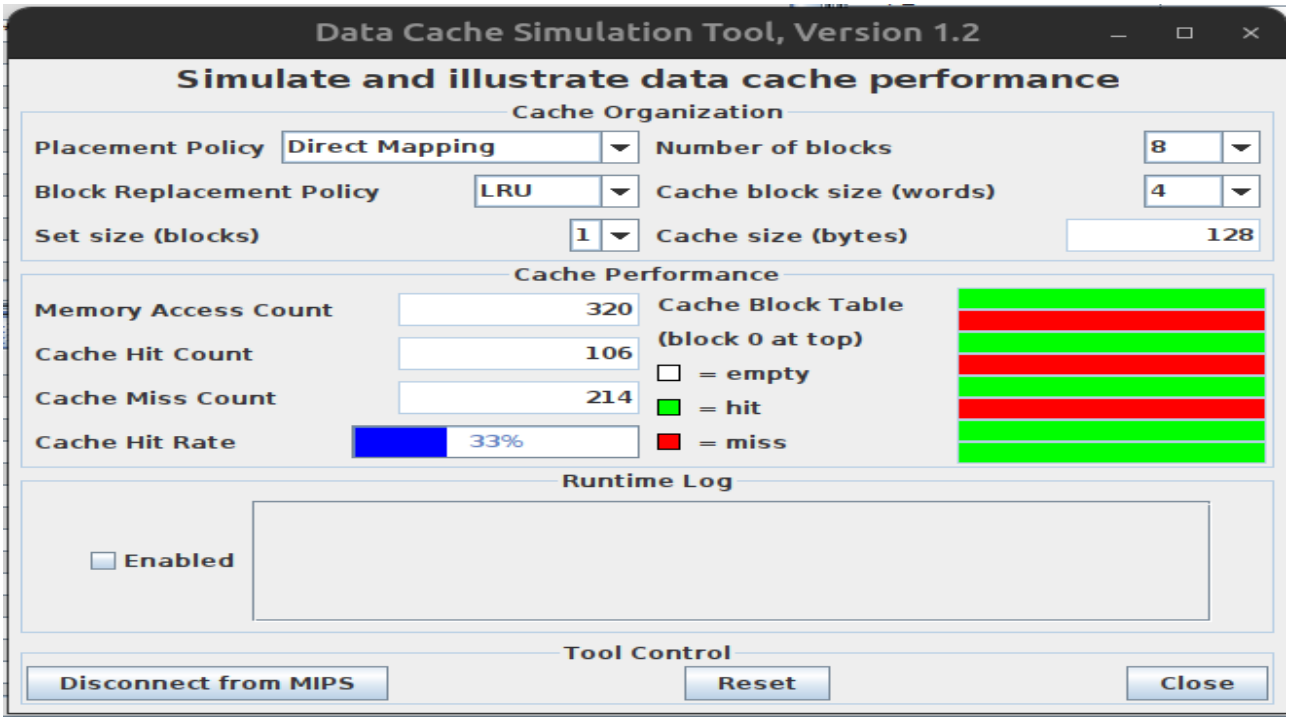
Ao iterações fracionadas pelo block size, o programa pode usar mais vezes o mesmo conjunto de blocos, aumentando a taxa de hits e evitando buscas em memórias mais lentas. Isso deve-se sobretudo à leitura de B ser transposta (iterando as linhas antes das colunas), fazendo com que blocos distantes de memória sejam necessários em sequência. O fracionamento reduz esse problema pois, embora as linhas sejam ainda sejam iteradas primeiro, será por menos vezes antes de começar a iteração por colunas, permitindo aproveitar mais um bloco já alocado na cache.

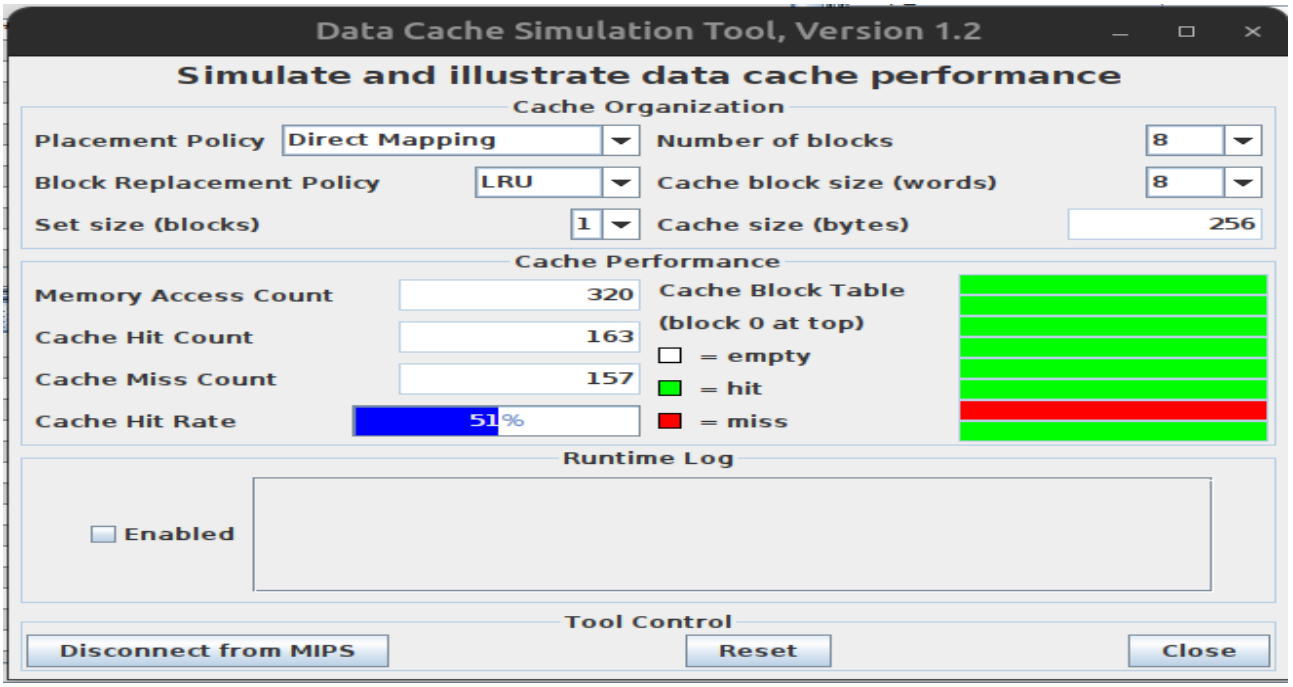
Código 1

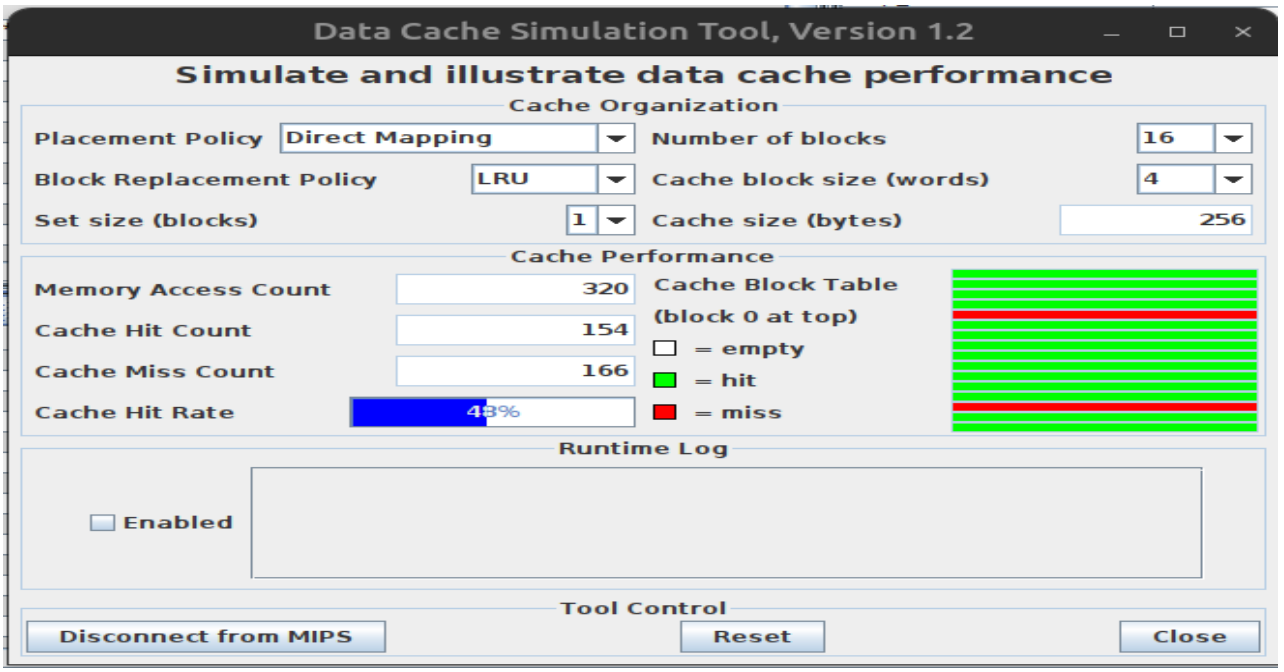
Tem mais acessos (320 - 192) devido a alocação prévia de valores na matriz (usada para conferir o resultado), que são realizadas iterando colunas primeiro, tanto para A quanto para B, fazendo com que o resultado seja influenciado positivamente

Os dois exemplos apontam que ter menos blocos, porém maiores, é mais efetivo, pois serão necessárias menos buscas por um endereço antes da cache encher e, consequentemente, menos falhas.

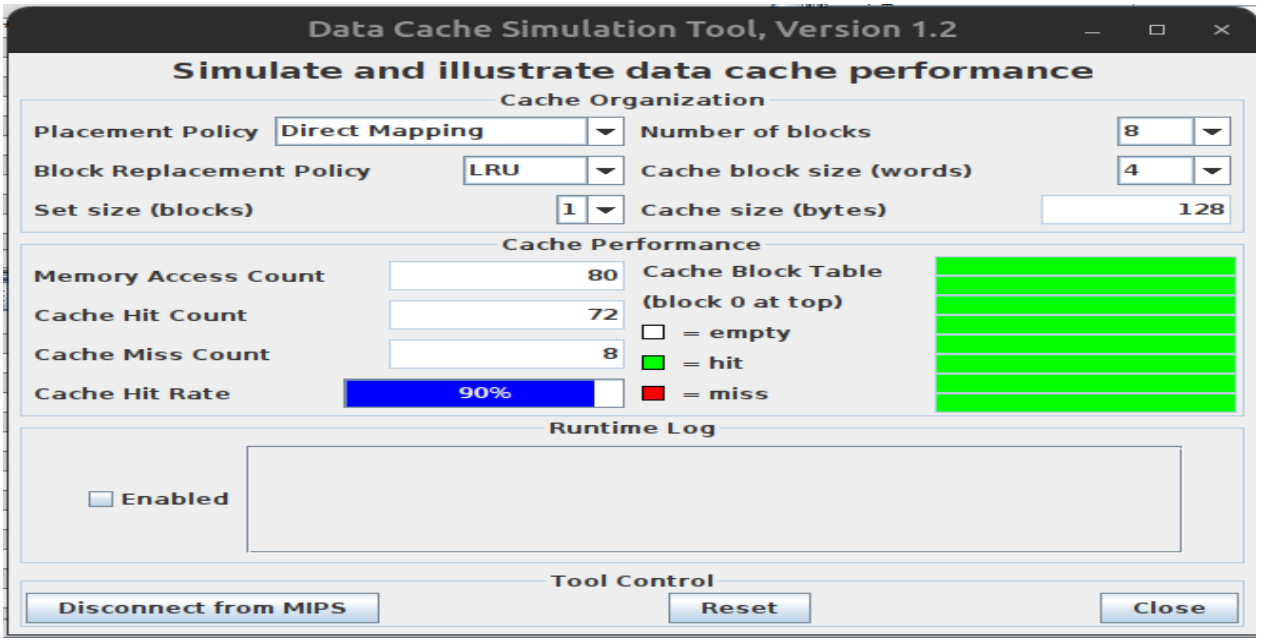
matriz 8x8

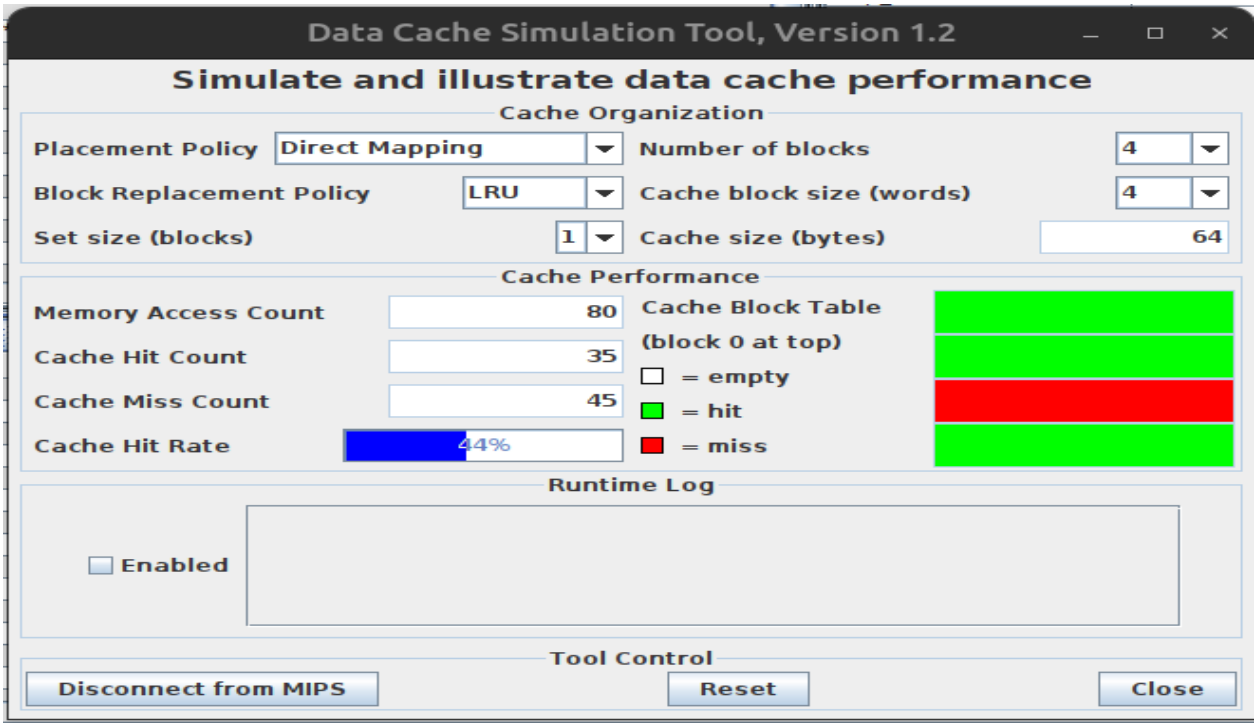


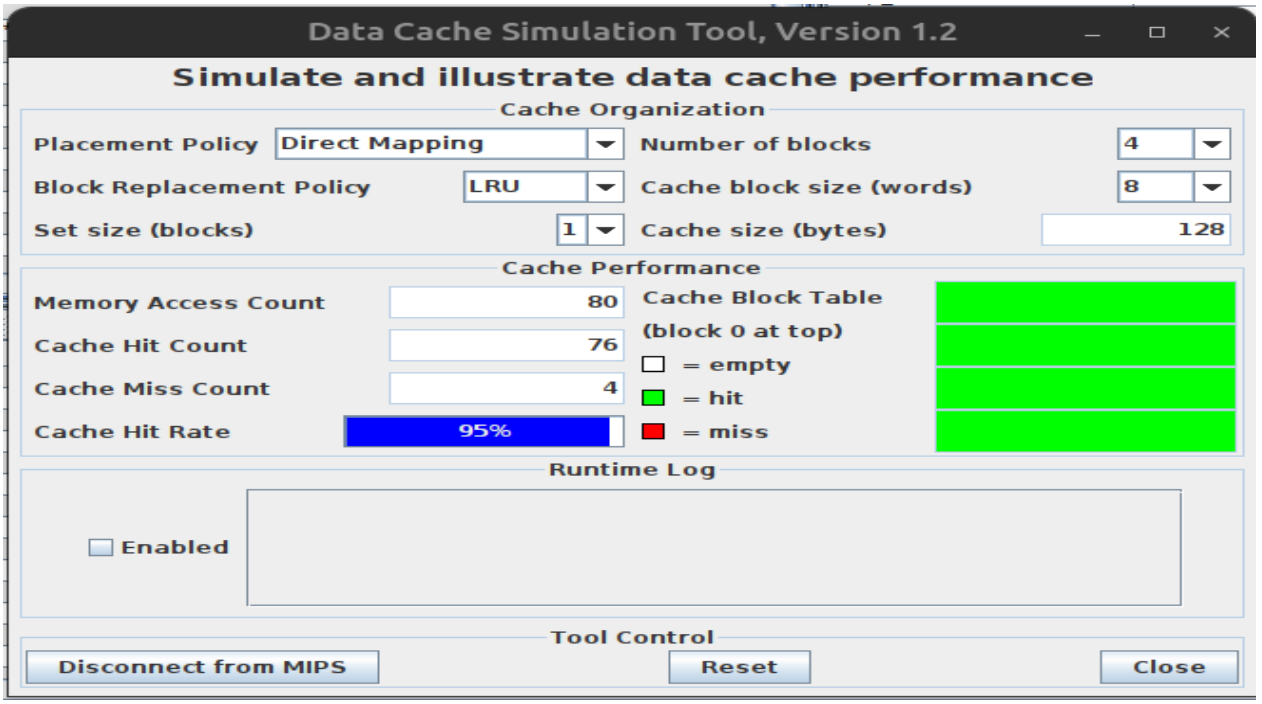



Data Cache Simulation Tool, Version 1.2
Simulate and illustrate data cache performance
Cache Organization
Placement Policy
Direct Mapping
Number of blocks
16
Block Replacement Policy
LRU
Cache block size (words)
4
Set size (blocks)
1
Cache size (bytes)
256
Cache Performance
Memory Access Count
320
Cache Block Table
(block 0 at top)
Cache Hit Count
154
= empty
Cache Miss Count
166
= hit
Cache Hit Rate
48%
= miss
Runtime Log
Enabled
Tool Control
Disconnect from MIPS
Reset
Close

matriz 4x4


Data Cache Simulation Tool, Version 1.2
Simulate and illustrate data cache performance
Cache Organization
Placement Policy
Direct Mapping
Number of blocks
8
Block Replacement Policy
LRU
Cache block size (words)
4
Set size (blocks)
1
Cache size (bytes)
128
Cache Performance
Memory Access Count
80
Cache Block Table
(block 0 at top)
Cache Hit Count
72
= empty
Cache Miss Count
8
= hit
Cache Hit Rate
90%
= miss
Runtime Log
Enabled
Tool Control
Disconnect from MIPS
Reset
Close



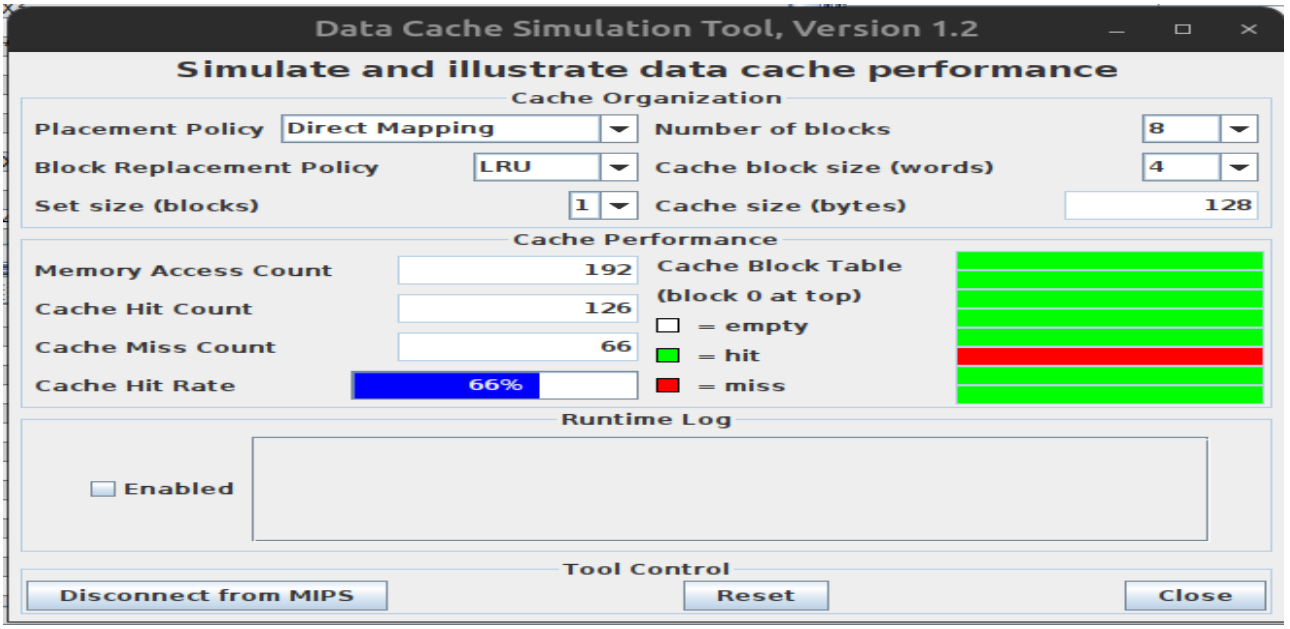


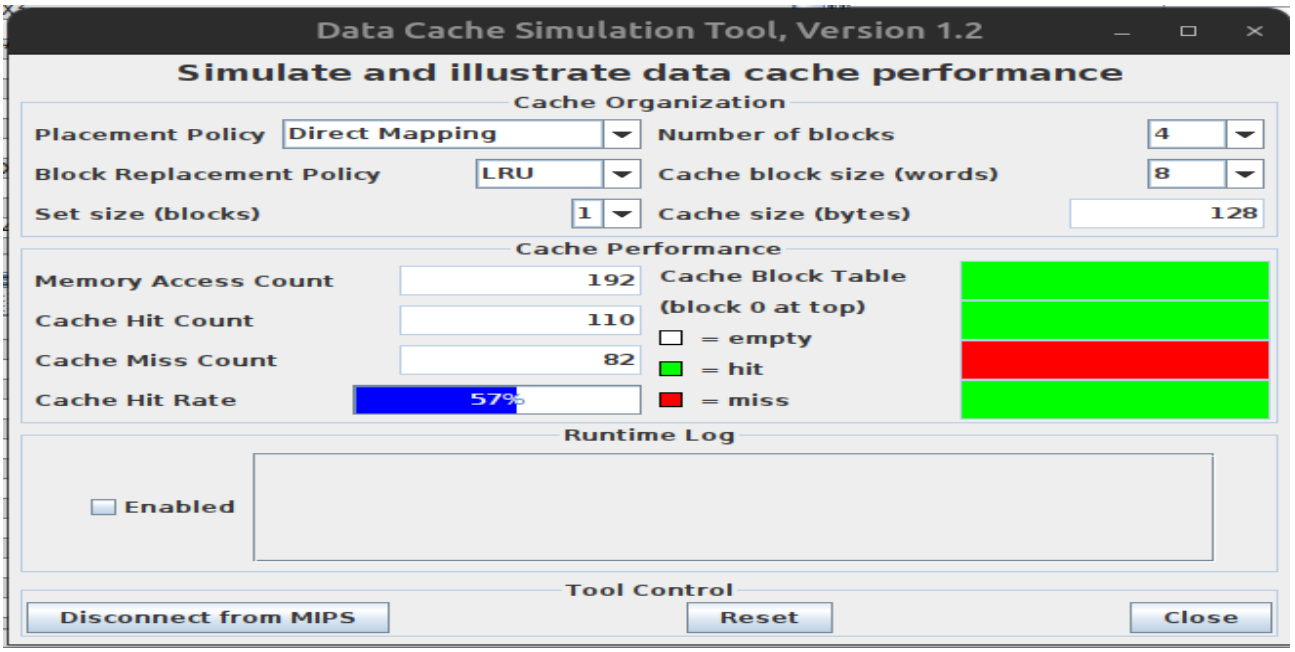
Código 2

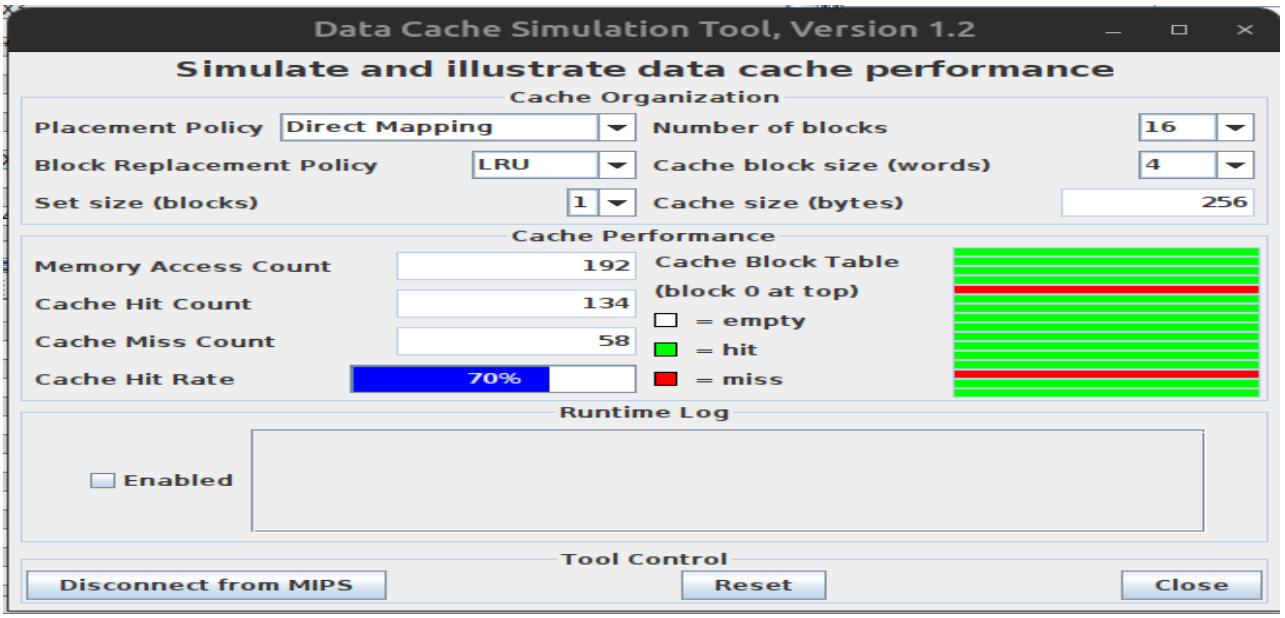
Tem menos acessos à memória, mas ainda sim consegue resultados melhores mesmo com matrizes e block sizes pequenos

Matriz 8X8, block size = 2

Este caso mostra a situação oposta à do exercício anterior, com mais blocos pequenos superando menos blocos grandes. Isso porque o segundo programa vai aproveitar uma parte maior de cada bloco, fazendo com que ter poucos blocos seja desvantajoso

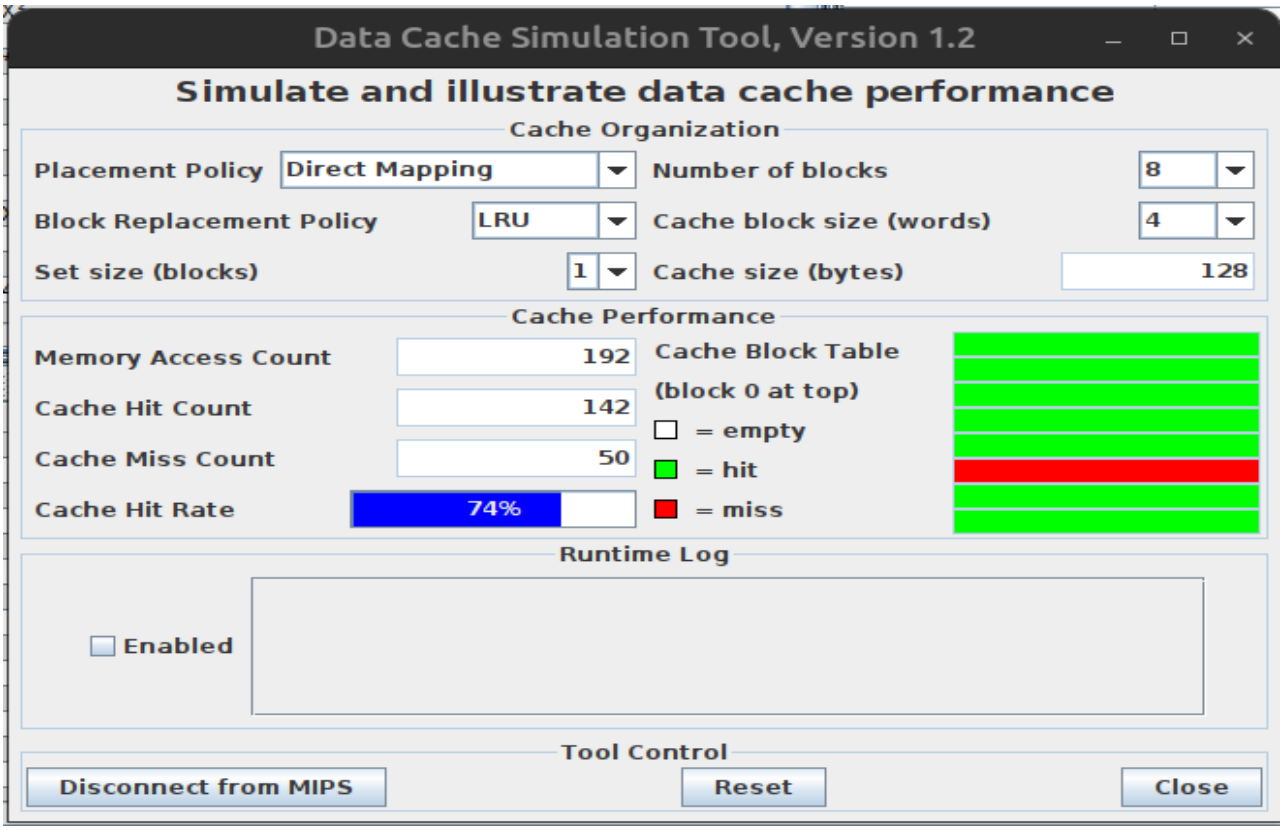


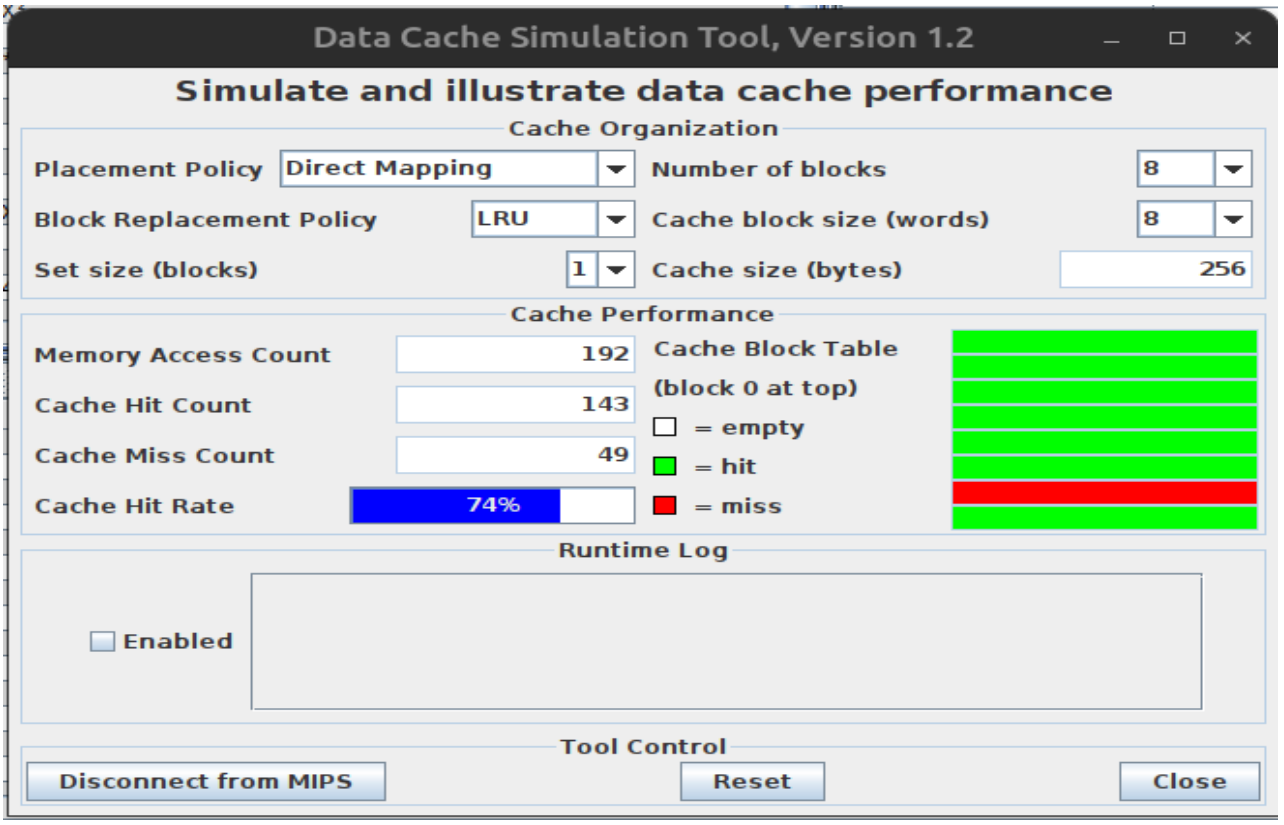




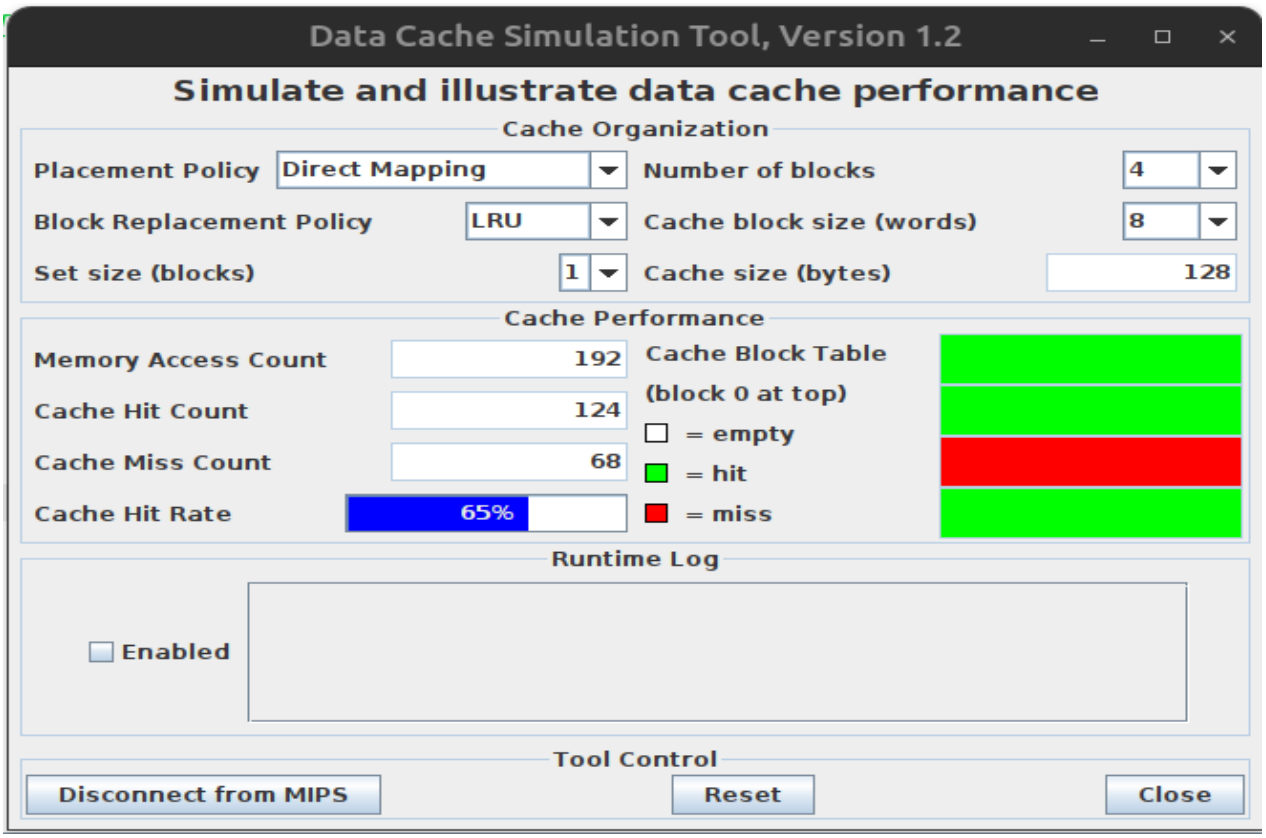
Matriz 8X8, block size = 4

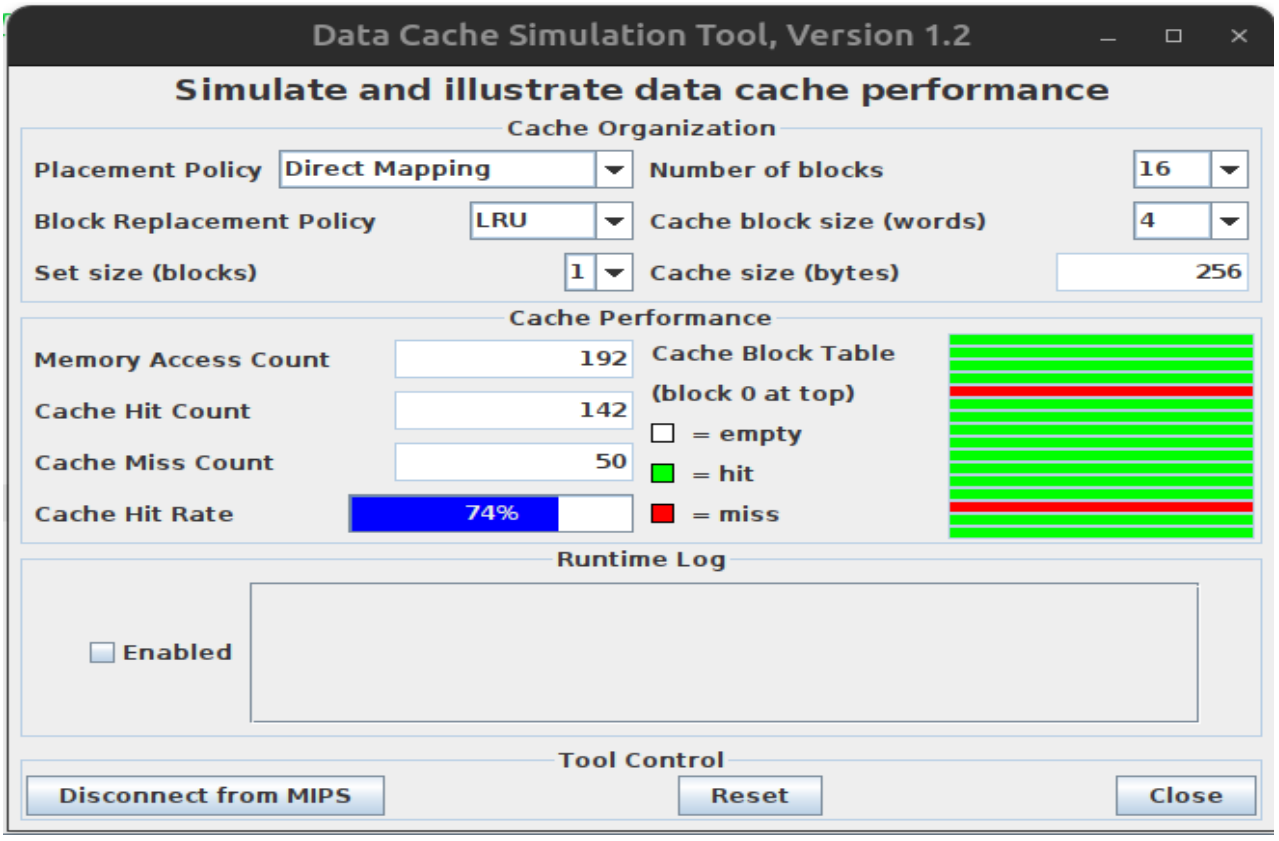
Como são aproveitadas 4 words de cada bloco, o ganho de desempenho de ir de 4 para 8 words/bloco foi nulo



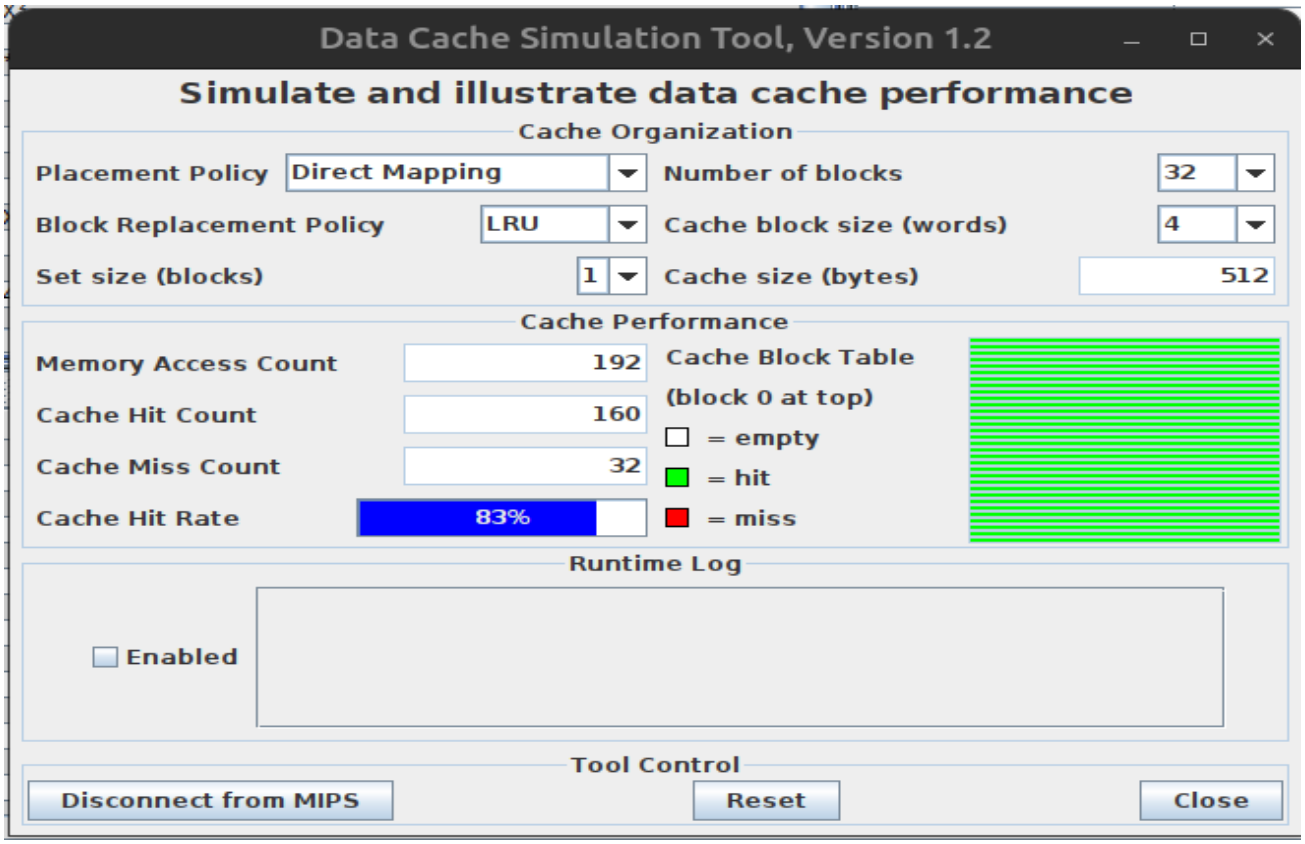


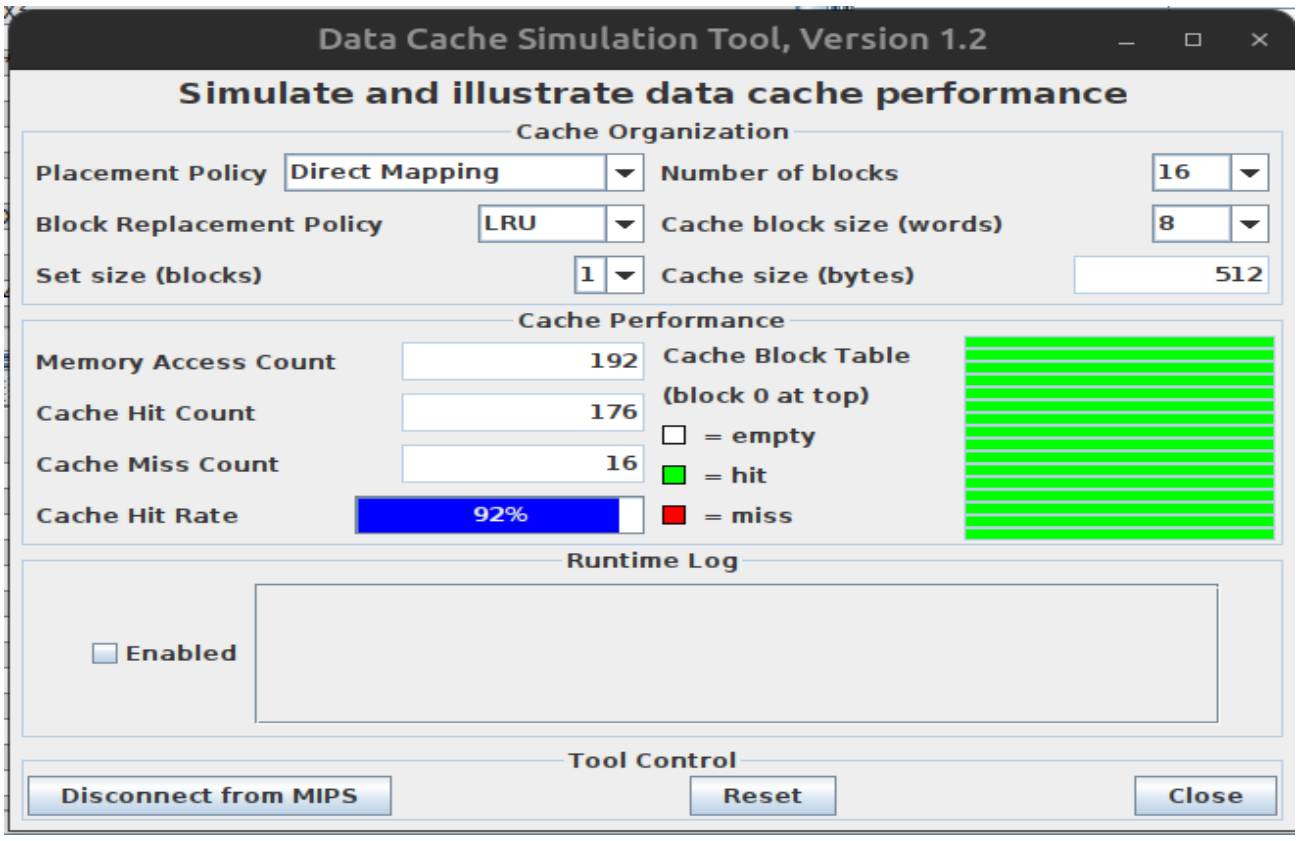
Porém, como o block size também é maior, meramente dobrar o número de blocos não é efetivo, pois não é grande o bastante para que os mesmos blocos possam permanecer após uma iteração das variáveis não-fracionadas (i, j), fazendo com que hajam mais trocas que compensam o ganho.





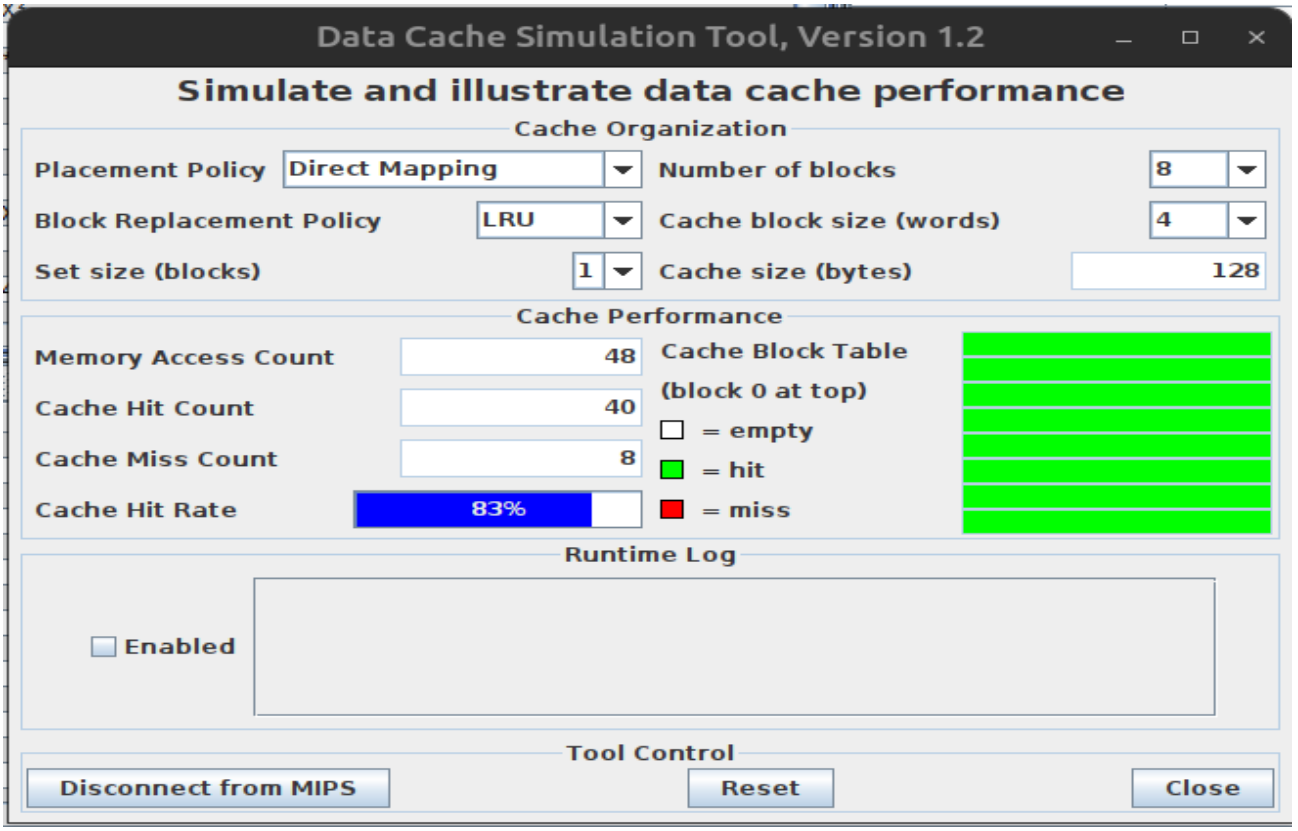
Entretanto, quadruplicar o tamanho da cache gera melhores resultados, especialmente quando se aumenta o tamanho de bloco (o efeito com 8 blocos e 16 words/bloco é maior ainda) pois os mesmos blocos poderão permanecer por mais tempo e ser usados mais vezes

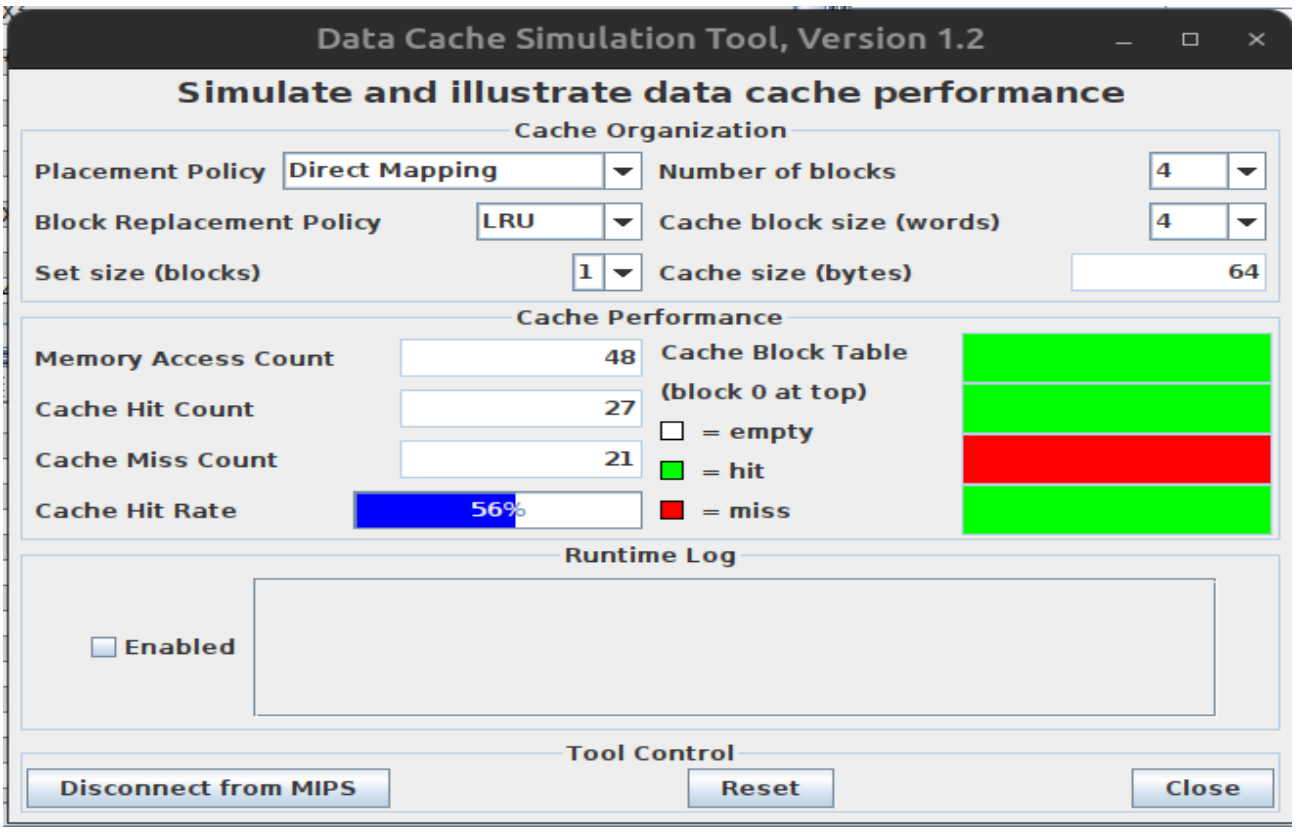




Matriz 4x4 block size = 2

Naturalmente, o efeito é menos perceptível com um matriz menor


Data Cache Simulation Tool, Version 1.2
Simulate and illustrate data cache performance
Cache Organization
Placement Policy
Direct Mapping
Number of blocks
8
Block Replacement Policy
LRU
Cache block size (words)
4
Set size (blocks)
1
Cache size (bytes)
128
Cache Performance
Memory Access Count
48
Cache Hit Count
40
Cache Miss Count
8
Cache Hit Rate
83%
Cache Block Table
(block 0 at top)
= empty
= hit
= miss
Runtime Log
Enabled
Tool Control
Disconnect from MIPS
Reset
Close


Data Cache Simulation Tool, Version 1.2
Simulate and illustrate data cache performance
Cache Organization
Placement Policy
Direct Mapping
Number of blocks
4
Block Replacement Policy
LRU
Cache block size (words)
4
Set size (blocks)
1
Cache size (bytes)
64
Cache Performance
Memory Access Count
48
Cache Hit Count
27
Cache Miss Count
21
Cache Hit Rate
56%
Cache Block Table
(block 0 at top)
= empty
= hit
= miss
Runtime Log
Enabled
Tool Control
Disconnect from MIPS
Reset
Close