Trabalho Prático 3 - Organização de Computadores

Gabriel Martins Miranda, João Carlos Ferraz, Luiz Felipe Gondim

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil
Matrículas: 2023028307, 2023087990, 2023028188

Introdução e Objetivos

Este trabalho prático da disciplina Organização de Computadores 1 teve como objetivo promover uma maior familiarização dos alunos com a hierarquia de memoria na cache. O objetivo do trabalho se dava em criar um simulador de memoria cache, demonstrando os HITS, MISSES e em qual linha da cache um bloco deveria ser armazenado.

1 Desenvolvimento e Testes

1.1 Desenvolvimento

O código criado simula uma memória cache com mapeamento associativo por conjunto, utilizando a política de substituição First In First Out (FIFO). Para isso, foi implementada a classe `SimuladorCache`, que, conforme esperado, simula a memória cache. Nela são definidos todos os parâmetros relevantes, desde a associatividade até o tamanho das linhas, além dos bits de offset, índice e outros.

Em seguida, temos a função auxiliar `obter_tag_grupo_identificador`, na qual o endereço hexadecimal é convertido para inteiro, são removidos os bits de offset e, então, são calculados a tag, o grupo e um identificador hexadecimal formatado.

Também é implementada a função `acessar_cache`, que, como o nome indica, realiza o acesso à cache dado um endereço hexadecimal. Nessa função, é feita a verificação para determinar se ocorreu um *MISS* ou um *HIT*, e caso necessário, realiza-se a substituição de linha. Além disso, cada acesso é registrado em um histórico. Por fim, há uma função para recuperar o histórico dos acessos a essa memória simulada.

1.2 Testes

Foram realizados um total de 10 testes para simular diversos casos possíveis, os 5 primeiros foram feitos com os parâmetros:

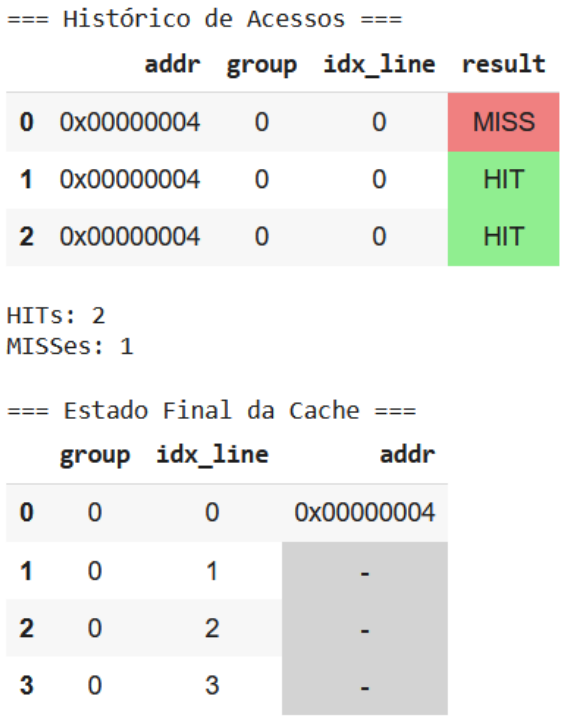
- cache-size = 4096
- line-size = 1024
- associativity = 4

Para os testes 6 e 7 a associatividade foi trocada para 1, simulando o mapeamento direto. E para os últimos 3 testes foram utilizados os seguintes parâmetros:

- cache-size = 8192
- line-size = 1024
- associativity = 2

1.2.1 Teste 1

Entrada: `'0x00001000', '0x00001000', '0x00001000'`
Resultado esperado nos acessos: MISS, HIT, HIT
Resultado esperado na cache: Apenas um bloco ocupado.



=== Histórico de Acessos ===

	addr	group	idx_line	result
0	0x00000004	0	0	MISS
1	0x00000004	0	0	HIT
2	0x00000004	0	0	HIT

HITs: 2
MISSes: 1

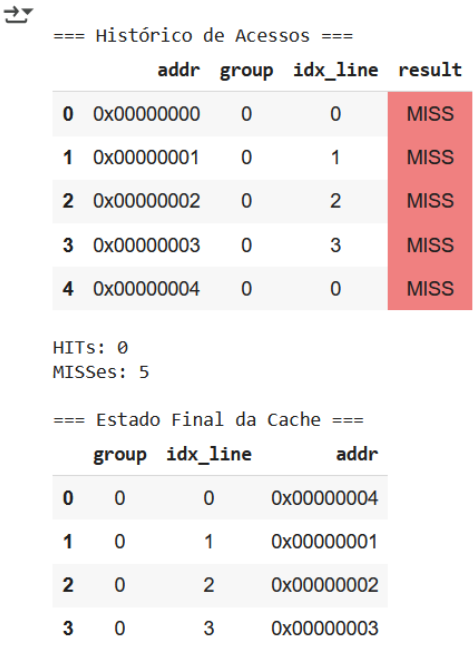
=== Estado Final da Cache ===

	group	idx_line	addr
0	0	0	0x00000004
1	0	1	-
2	0	2	-
3	0	3	-

Figura 1: Estado da cache após o Teste 1

1.2.2 Teste 2

Entrada: `'0x00000000', '0x00000400', '0x00000800', '0x00000C00', '0x00001000'`
Resultado esperado nos acessos: MISS, MISS, MISS, MISS, MISS
Resultado esperado na cache: Quatro blocos ocupados, com o bloco referente a '0x00000000' sendo substituído.



=== Histórico de Acessos ===

	addr	group	idx_line	result
0	0x00000000	0	0	MISS
1	0x00000001	0	1	MISS
2	0x00000002	0	2	MISS
3	0x00000003	0	3	MISS
4	0x00000004	0	0	MISS

HITs: 0
MISSes: 5

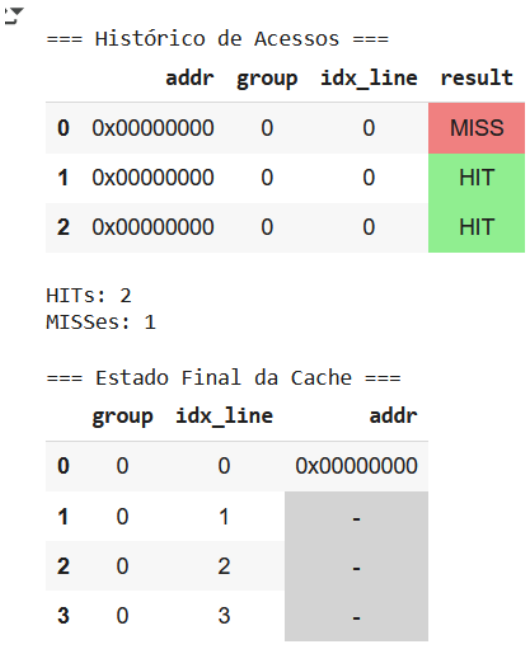
=== Estado Final da Cache ===

	group	idx_line	addr
0	0	0	0x00000004
1	0	1	0x00000001
2	0	2	0x00000002
3	0	3	0x00000003

Figura 2: Estado da cache após o Teste 2

1.2.3 Teste 3

Entrada: `'0x00000001', '0x000003FF', '0x00000010'`
Resultado esperado nos acessos: MISS, HIT, HIT
Resultado esperado na cache: Apenas um bloco ocupado.



=== Histórico de Acessos ===

	addr	group	idx_line	result
0	0x00000000	0	0	MISS
1	0x00000000	0	0	HIT
2	0x00000000	0	0	HIT

HITs: 2
MISSes: 1

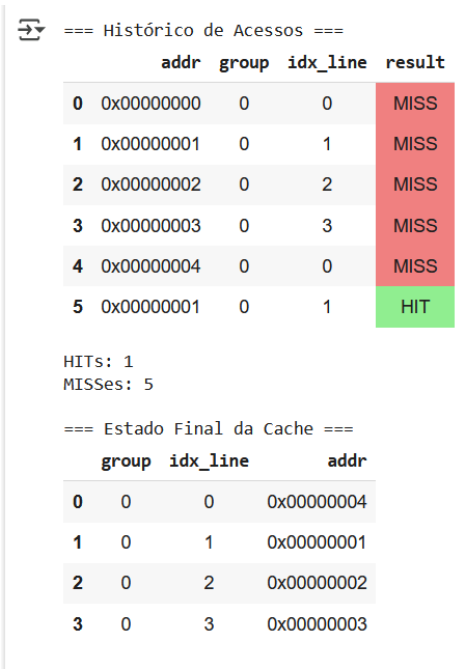
=== Estado Final da Cache ===

	group	idx_line	addr
0	0	0	0x00000000
1	0	1	-
2	0	2	-
3	0	3	-

Figura 3: Estado da cache após o Teste 3

1.2.4 Teste 4

Entrada: `'0x00000000', '0x00000400', '0x00000800', '0x00000C00', '0x00001000', '0x00000400'`
Resultado esperado nos acessos: MISS, MISS, MISS, MISS, MISS, HIT
Resultado esperado na cache: Quatro blocos ocupados, com o bloco referente a '0x00000000' sendo substituído.



```
=== Histórico de Acessos ===
```

	addr	group	idx_line	result
0	0x00000000	0	0	MISS
1	0x00000001	0	1	MISS
2	0x00000002	0	2	MISS
3	0x00000003	0	3	MISS
4	0x00000004	0	0	MISS
5	0x00000001	0	1	HIT

```
HITs: 1
MISSes: 5

=== Estado Final da Cache ===
```

	group	idx_line	addr
0	0	0	0x00000004
1	0	1	0x00000001
2	0	2	0x00000002
3	0	3	0x00000003

Figura 4: Estado da cache após o Teste 4

1.2.5 Teste 5

Entrada: `'0x00000000', '0x00000400', '0x00000000', '0x00000400', '0x00000800', '0x00000C00', '0`
Resultado esperado nos acessos: MISS, MISS, HIT, HIT, MISS, MISS, MISS
Resultado esperado na cache: Quatro blocos ocupados, com o bloco referente a '0x00000000' sendo substituído.

	addr	group	idx_line	result
0	0x00000000	0	0	MISS
1	0x00000001	0	1	MISS
2	0x00000000	0	0	HIT
3	0x00000001	0	1	HIT
4	0x00000002	0	2	MISS
5	0x00000003	0	3	MISS
6	0x00000004	0	0	MISS

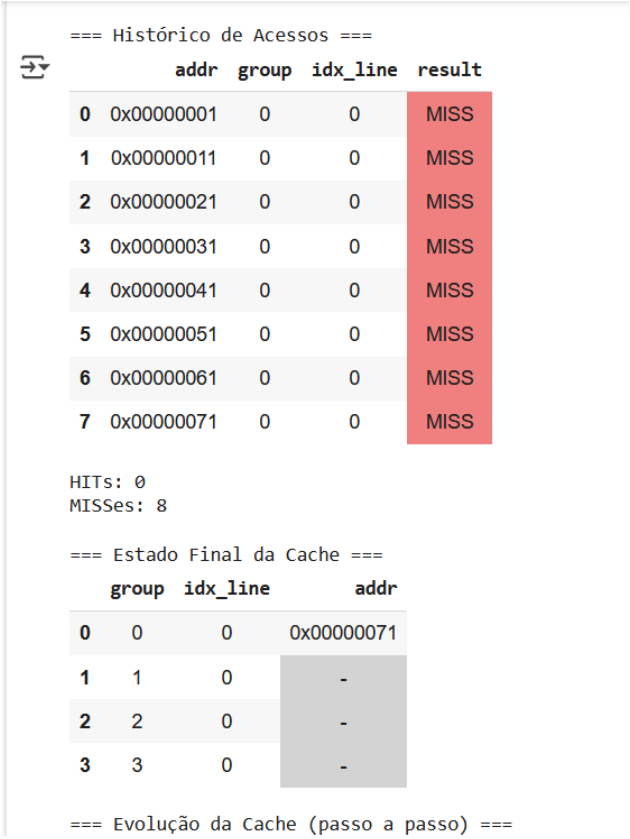
```
HITs: 2
MISSes: 5

=== Estado Final da Cache ===
```

	group	idx_line	addr
0	0	0	0x00000004
1	0	1	0x00000001
2	0	2	0x00000002
3	0	3	0x00000003

Figura 5: Estado da cache após o Teste 5

1.2.6 Teste 6

Entrada: `'0x00001000', '0x00011000', '0x00021000', '0x00031000' '0x00041000', '0x00051000', '0x00061000', '0x00071000'`
Resultado esperado nos acessos: MISS, MISS, MISS, MISS, MISS, MISS, MISS, MISS
Resultado esperado na memória: Apenas um bloco permanece armazenado ao final ('0x00071000'), pois todos os acessos mapeiam para o mesmo conjunto e causam substituição a cada nova leitura.



=== Histórico de Acessos ===

	addr	group	idx_line	result
0	0x00000001	0	0	MISS
1	0x00000011	0	0	MISS
2	0x00000021	0	0	MISS
3	0x00000031	0	0	MISS
4	0x00000041	0	0	MISS
5	0x00000051	0	0	MISS
6	0x00000061	0	0	MISS
7	0x00000071	0	0	MISS

HITs: 0
MISSes: 8

=== Estado Final da Cache ===

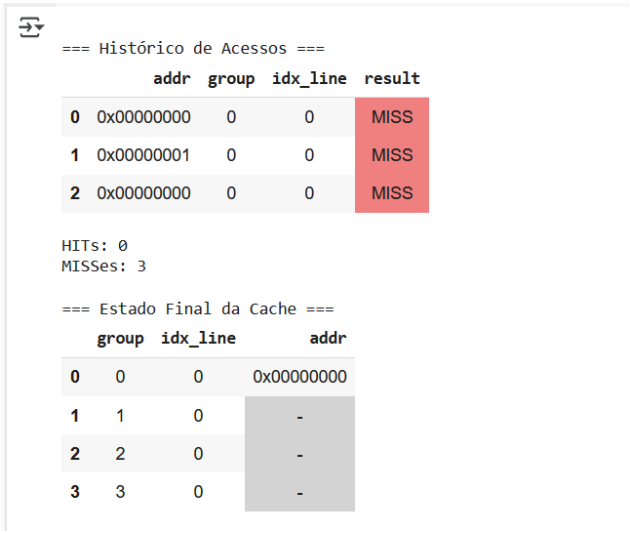
	group	idx_line	addr
0	0	0	0x00000071
1	1	0	-
2	2	0	-
3	3	0	-

=== Evolução da Cache (passo a passo) ===

Figura 6: Estado da cache após o Teste 6

1.2.7 Teste 7

Entrada: `'0x00000000', '0x00001000', '0x00000000'`
Resultado esperado nos acessos: MISS, MISS, MISS
Resultado esperado na memória: Apenas o bloco '0x00000000' permanece ao final, tendo sido substituído temporariamente por '0x00001000' no segundo acesso.



=== Histórico de Acessos ===

	addr	group	idx_line	result
0	0x00000000	0	0	MISS
1	0x00000001	0	0	MISS
2	0x00000000	0	0	MISS

HITs: 0
MISSes: 3

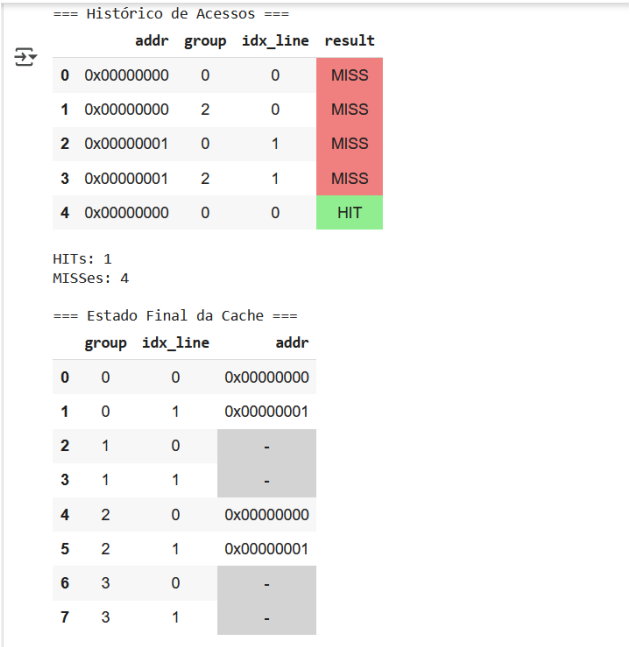
=== Estado Final da Cache ===

	group	idx_line	addr
0	0	0	0x00000000
1	1	0	-
2	2	0	-
3	3	0	-

Figura 7: Estado da cache após o Teste 7

1.2.8 Teste 8

Entrada: `'0x00000000', '0x00000800', '0x00001000', '0x00001800', '0x00000000'`
Resultado esperado nos acessos: MISS, MISS, MISS, MISS, HIT
Resultado esperado na memória: Quatro blocos armazenados, com o bloco '0x00000000' ainda presente no conjunto ao final do teste, possibilitando o HIT no último acesso.



=== Histórico de Acessos ===

	addr	group	idx_line	result
0	0x00000000	0	0	MISS
1	0x00000000	2	0	MISS
2	0x00000001	0	1	MISS
3	0x00000001	2	1	MISS
4	0x00000000	0	0	HIT

HITs: 1
MISSes: 4

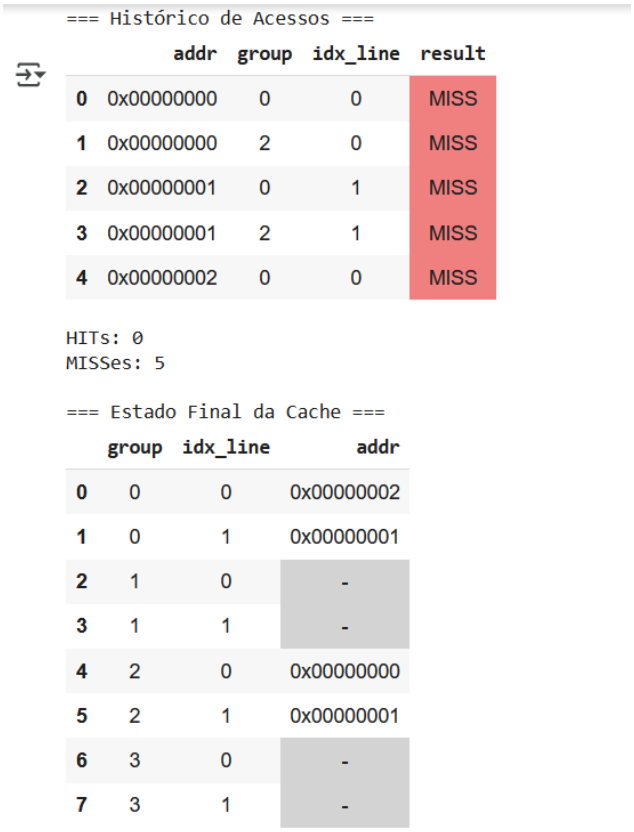
=== Estado Final da Cache ===

	group	idx_line	addr
0	0	0	0x00000000
1	0	1	0x00000001
2	1	0	-
3	1	1	-
4	2	0	0x00000000
5	2	1	0x00000001
6	3	0	-
7	3	1	-

Figura 8: Estado da cache após o Teste 8

1.2.9 Teste 9

Entrada: `'0x00000000', '0x00000800', '0x00001000', '0x00001800', '0x00002000'`
Resultado esperado nos acessos: MISS, MISS, MISS, MISS, MISS
Resultado esperado na memória: Quatro blocos armazenados, com substituição ocorrendo no último acesso ('0x00002000'), pois o conjunto atingiu sua capacidade máxima de 2 linhas.



```
=== Histórico de Acessos ===
```

	addr	group	idx_line	result
0	0x00000000	0	0	MISS
1	0x00000000	2	0	MISS
2	0x00000001	0	1	MISS
3	0x00000001	2	1	MISS
4	0x00000002	0	0	MISS

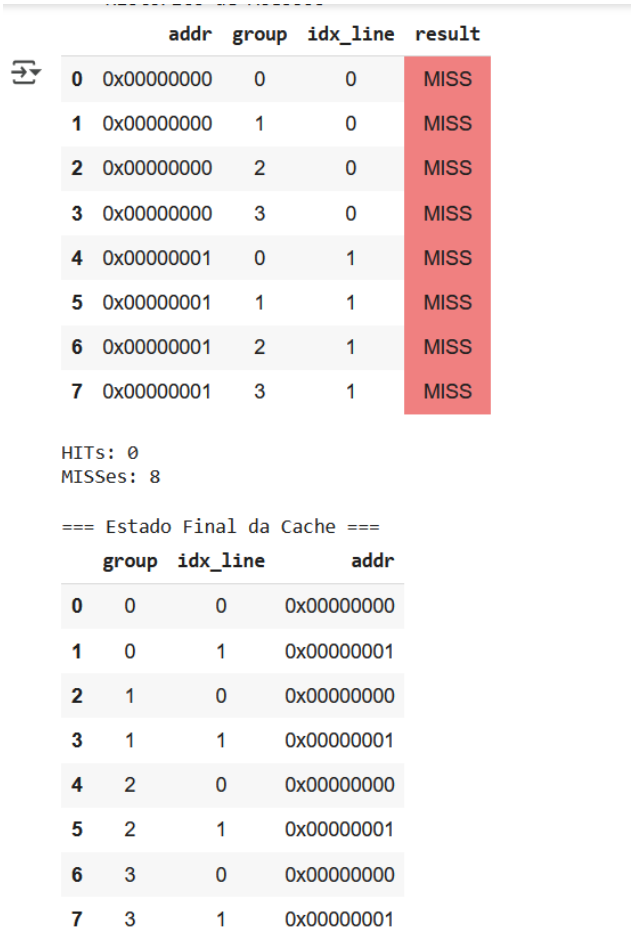
```
HITs: 0
MISSes: 5

=== Estado Final da Cache ===
```

	group	idx_line	addr
0	0	0	0x00000002
1	0	1	0x00000001
2	1	0	-
3	1	1	-
4	2	0	0x00000000
5	2	1	0x00000001
6	3	0	-
7	3	1	-

Figura 9: Estado da cache após o Teste 9

1.2.10 Teste 10

Entrada: `'0x00000000', '0x00000400', '0x00000800', '0x00000C00'`
`'0x00001000', '0x00001400', '0x00001800', '0x00001C00'`
Resultado esperado nos acessos: MISS, MISS, MISS, MISS, MISS, MISS, MISS, MISS
Resultado esperado na memória: Oito blocos distintos armazenados, ocupando completamente todos os conjuntos e linhas disponíveis, sem substituições ao final da sequência.



	addr	group	idx_line	result
0	0x00000000	0	0	MISS
1	0x00000000	1	0	MISS
2	0x00000000	2	0	MISS
3	0x00000000	3	0	MISS
4	0x00000001	0	1	MISS
5	0x00000001	1	1	MISS
6	0x00000001	2	1	MISS
7	0x00000001	3	1	MISS

```
HITs: 0
MISSes: 8

=== Estado Final da Cache ===
```

	group	idx_line	addr
0	0	0	0x00000000
1	0	1	0x00000001
2	1	0	0x00000000
3	1	1	0x00000001
4	2	0	0x00000000
5	2	1	0x00000001
6	3	0	0x00000000
7	3	1	0x00000001

Figura 10: Estado da cache após o Teste 10

1.3 Uso de Inteligência Artificial como ferramenta

Nosso grupo fez uso de algumas IA's conhecidas como ChatGPT e Gemini para gerar casos de teste, que foram usados para testar a robustez do nosso código. Além disso, foi fonte de consulta para procurar saber mais sobre algumas bibliotecas específicas do python como a "deque"para a implementação da fila.

2 Conclusão

Em suma, esse trabalho foi importante para elevar o entendimento dos integrantes do grupo acerca do funcionamento geral de uma memória cache, assumindo assim um papel de extrema importancia dentro do que a matéria organização de computadores se propõe, que é formar estudantes capazes de lidar e entender esse tipo de estrutura de hardware.