Simulador de Mapeamento de Memória

Introdução

O princípio básico da memória cache está em manter cópia de instruções que são recentemente acessadas, evitando que o processador precise buscar esta informação nas estruturas da memória principal. Qualquer dado pode ser trazido para a memória cache, garantindo um alto ganha de desempenho. Baseado nesta tarefa, este simulador foi desenvolvido com o objetivo de demonstrar todo o funcionamento envolvido nas estratégias de mapeamento de memória, determinando o caminho de cada dado entre a memória principal e a cache. O simulador de mapeamento foi desenvolvido para visualizar cada etapa de cópia e acesso de endereços e exibir as taxas de hit e miss para cada um dos tipos de mapeamentos inseridos no modelo. Este simulador foi totalmente desenvolvido em *Python3.6* e executado diretamente em computadores Windows por meio do prompt de comando.

Funcionamento do Script

O funcionamento do script consiste em receber valores com o tamanho da memória principal, memória cache, bit de palavra e o tipo de mapeamento desejado. Basicamente, o script tenta construir a arquitetura da memória cache. Para isso, foram desenvolvidos `arrays` para simular a estrutura da cache, incluindo os campos de `index, tag, bit validador` e um campo para armazenar os dados transferidos dos blocos da memória principal. A execução do script pode ser feita utilizando da seguinte maneira:

- Após descompactar o `.zip` com os arquivos de teste e também os scripts `run.py` e `mapping.py`, realize a execução do seguinte comando no diretório raiz, seja no prompt de comando no Windows ou terminal Linux.
- Execução: `python run.py -local C:\Teste\ProjetoOC -map direto`

Argumentos

Para exibir as opções possíveis de comando, basta apenas digitar `-h` ou `--help` após o comando de execução do script, ou seja, `python run.py -h`. Os principais comandos disponíveis no simulador podem ser conferidos a seguir:

- `-h, --help`, show this help message and exit.
- `-show`, este argumento indica o tipo de visualização do simulador. Caso o valor este atribuído como `yes` ou `y`, o simulador irá exibir todo conteúdo da memória cache em cada uma das inserções e acessos. Caso o valor seja `no` ou `n`, o simulador irá apenas apresentar a fração de acertos e erros, incluindo a configuração escolhida. Por padrão, a visualização completa está ativa.
- `-mp`, este argumento indica a capacidade total em bytes da memória principal. Por padrão, este valor foi definido como sendo 4096 bytes. Para facilitar o uso do simulador e visualização das etapas de substituição, todos os valores foram definidos em bytes.
- `-cache`, este argumento indica a capacidade total em bytes da memória cache. Por padrão, este valor foi definido como sendo 32 bytes.
- `-w`, este argumento indica a capacidade em bits utilizada para representar o campo de palavras (offset bit), os últimos bits da instrução.
- `-map`, este argumento indica o tipo de mapeamento desejado. As opções disponíveis são representadas da seguinte maneira, `fully-associative` para o mapeamento

associativo, `set-associative` para o associativo por conjunto e direto para utilizar o mapeamento `direto`. Além disso, por padrão, o mapeamento associativo por conjunto foi construído com apenas dois conjuntos denominados `set1` e `set2`.

- `-sub`, este argumento indica o tipo de política de substituição. Por padrão, este campo está definido como `-all`, que faz com que o script execute o mapeamento escolhido para todas as opções válidas de políticas de substituição. As opções disponíveis são representadas da seguinte maneira: `FIFO, LRU, LFU` ou `Random`.
- `-local`, este campo é obrigatório e deve ser preenchido corretamente com o endereço do arquivo. O arquivo deve estar no formato `.txt` e cada linha deve conter o valor em decimal dos endereços de memória, já que o script realiza automaticamente a conversão para binário.

- **Modo de geração automática:** Este é um modo padrão do simulador que gera automaticamente todos os cenários possíveis, com base nos tipos de mapeamento e políticas definidas como padrão para o script. Neste caso em especial, só é necessário definir o local de arquivo de testes, ou outros valores para a memória, cache e bit de palavra, caso seja necessário. Se tais valores não forem informados, o simulador irá executar os comandos com o campo padrão pré-estabelecido. Com isso, a *memória principal = 4096 bytes*, *a cache = 32 bits* e o *bit (offset) = 2*. Segue abaixo o comando de exemplo para gerar automaticamente os cenários:

```
python run.py -map all -sub all -local teste.txt
```

- Finalmente, com todos os argumentos definidos, considere o seguinte comando para execução do script, onde, o valor de memória cache é 16 bytes, a memória principal possui 4096 bytes, o *offset* bit possui 2 bits na tabela de instrução e o mapeamento é direto (resultados apresentados na Figura 1).

```
python run.py -mp 4096 -cache 16 -w 2 -map fully-associative -sub FIFO -
local C:\Users\PROJETOOC\teste.txt -show yes
```

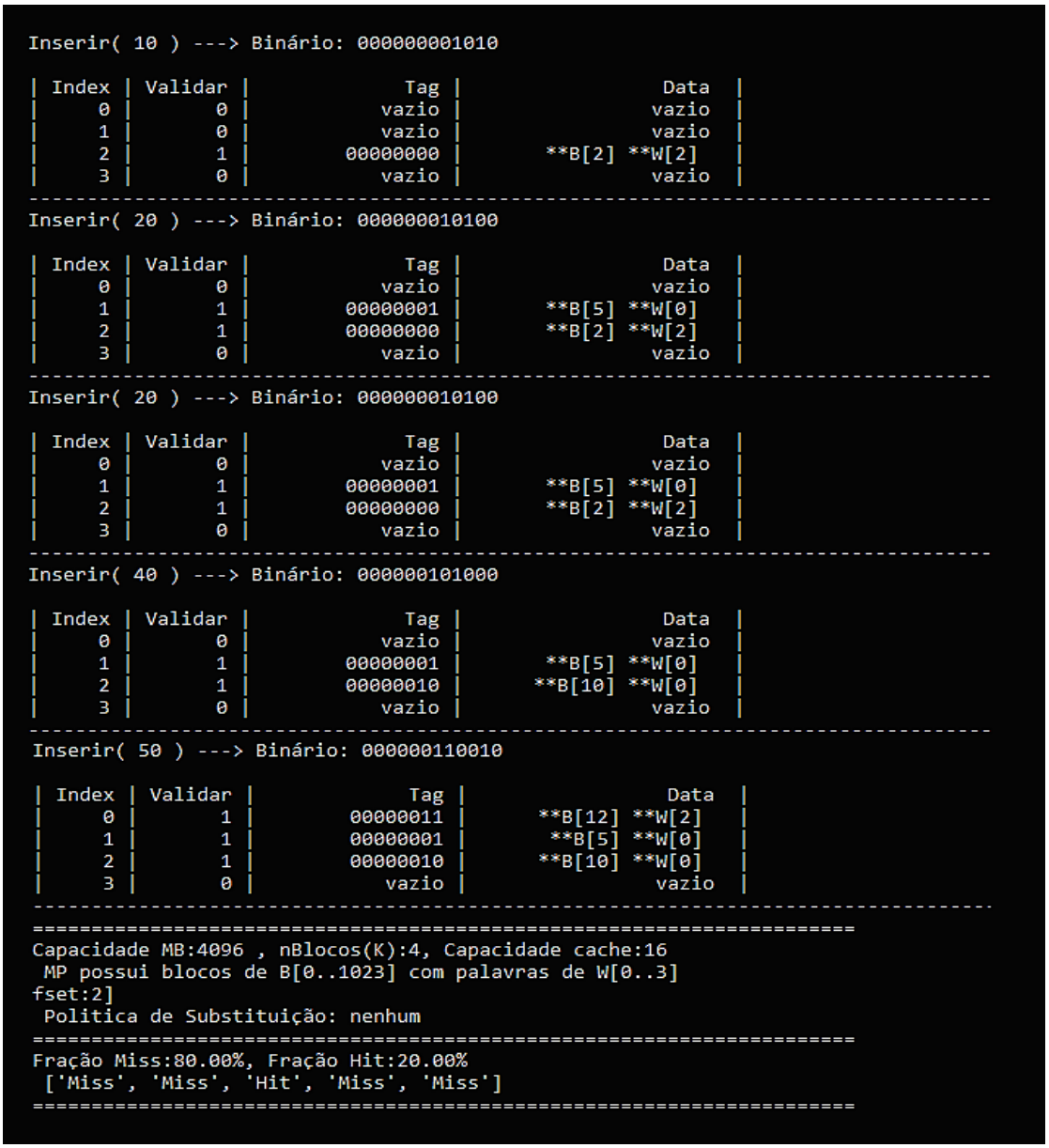

Figura 1 - Teste de Execução 1

Mapeamento

Os valores podem ser inseridos em formato decimal em um arquivo de texto simples. Na verdade, antes de qualquer operação, o simulador realiza a transição de qualquer valor recebido para o formato binário, conforme a quantidade de bits de cada instrução. Após o script obter todos os dados e também os valores de entrada, tais como capacidade da memória principal, da cache e também do *offset bit*, todo cálculo é realizado para definir o tamanho dos campos de index, tag, set. Seguindo este conceito, os dados recebidos são armazenados com base na política de substituição e regras para cada tipo de mapeamento selecionado. Para facilitar este processo, a cache e a memória principal são representadas por arrays multidimensionais, onde cada linha ou coluna representa um determinado campo, como se pode observar por meio da Figura 2.

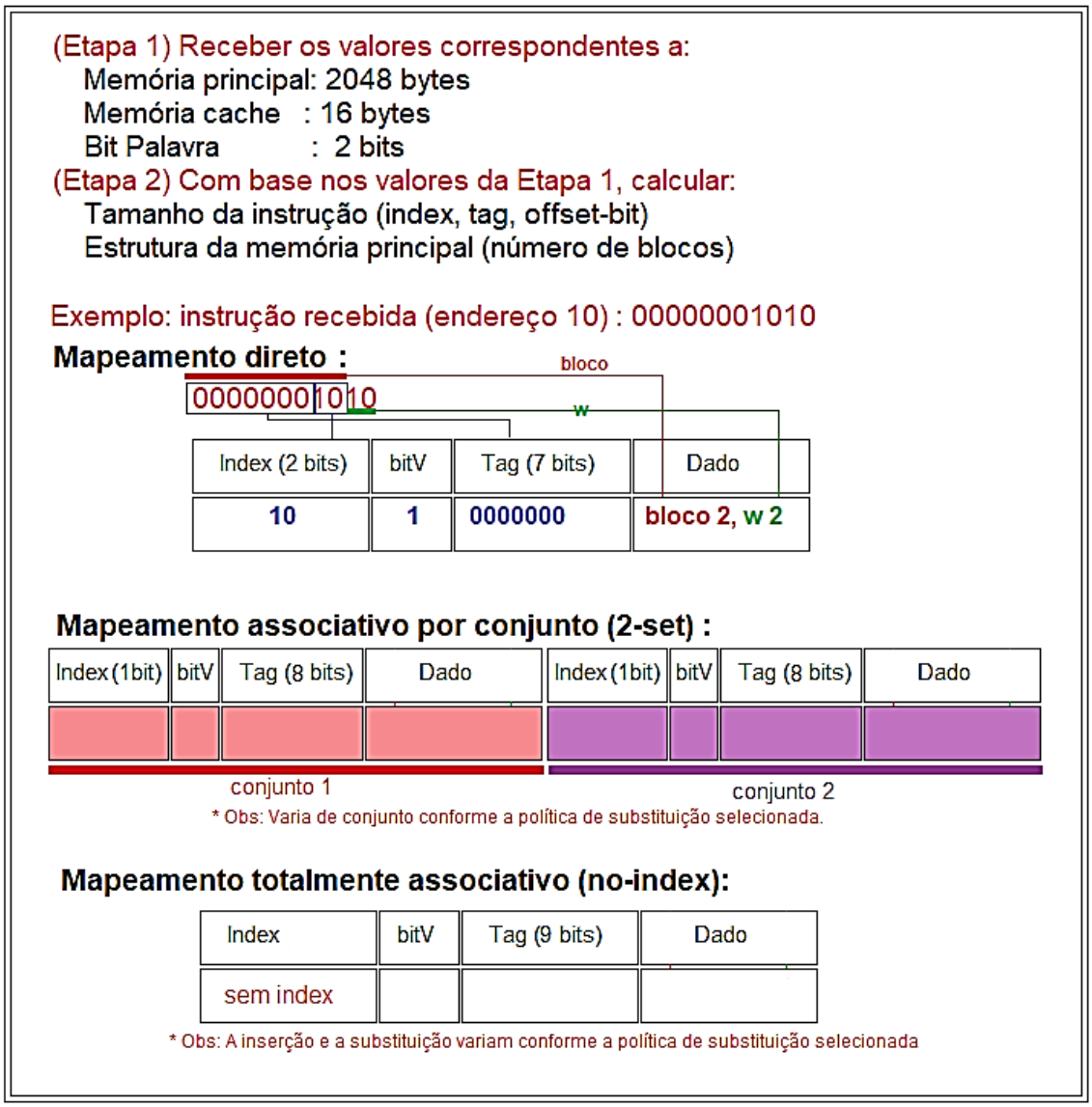

Figura 2 - Algoritmo de Mapeamento

Tipos de Mapeamento

Mapeamento Direto: É a forma de implementação mais simples, onde a posição da palavra na memória cache depende exclusivamente do endereço da palavra na memória principal. Neste caso, o campo de `index` no endereço recebido é o mesmo da posição `index` da cache (em binário). Se o `bit validador` é igual a 1, isso quer dizer que cache já está ocupada. Então é necessário verificar o campo `tag` da instrução recebida. Caso a `tag` seja simular em ambas as instruções, temos um `cache hit`, caso contrário um `cache miss` (Figura 3).

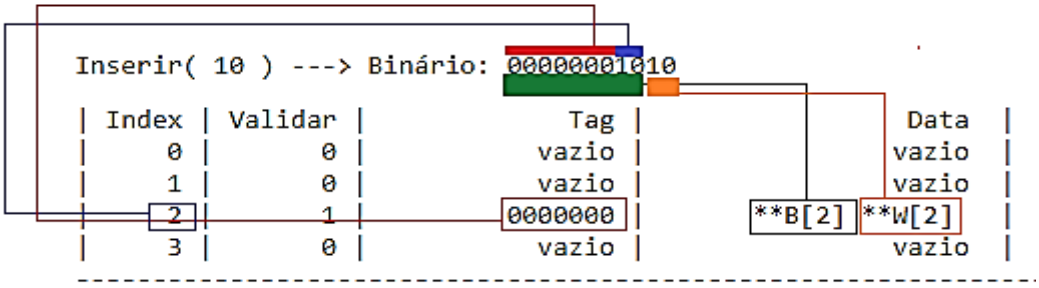

Figura 3 - Como funciona o Mapeamento Direto no Simulador

Mapeamento Totalmente Associativo: Este modelo tenta resolver conflitos e limitações do mapeamento direto. Nele, qualquer endereço pode ser mapeado em qualquer posição da cache. Para isso, é necessário realizar uma busca em todas as posições para verificar se a `tag` já é um conteúdo da cache, se verdadeiro, então temos um hit. Nesta parte, o script desenvolvido percorre todo vetor em busca de um elemento semelhante (hit), caso contrário, a instrução é armazenada em qualquer posição (sem index, depende totalmente da política de substituição selecionada).

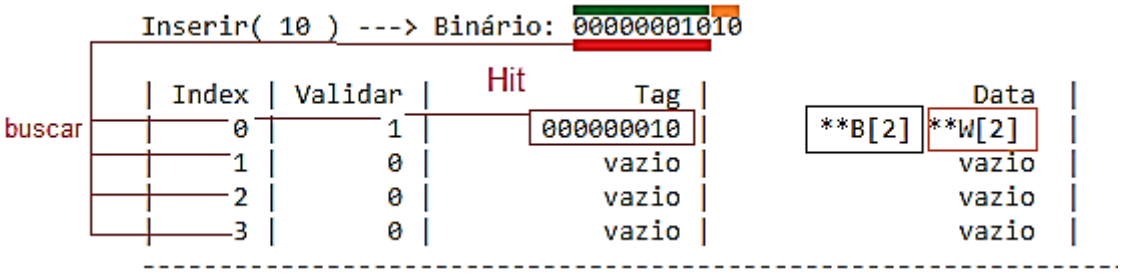

Figura 4 - Como funciona o Mapeamento Totalmente Associativo no Simulador

Mapeamento Associativo por Conjunto: Este modelo é simular ao totalmente associativo, porém a cache está dividida em conjuntos. Nele, o endereço deve ser inserido com base na posição de `index` do `array` principal. Com isso, a implementação foi realizada focando em dois conjuntos associativos de cache (2-set), conforme demonstra o exemplo a seguir.

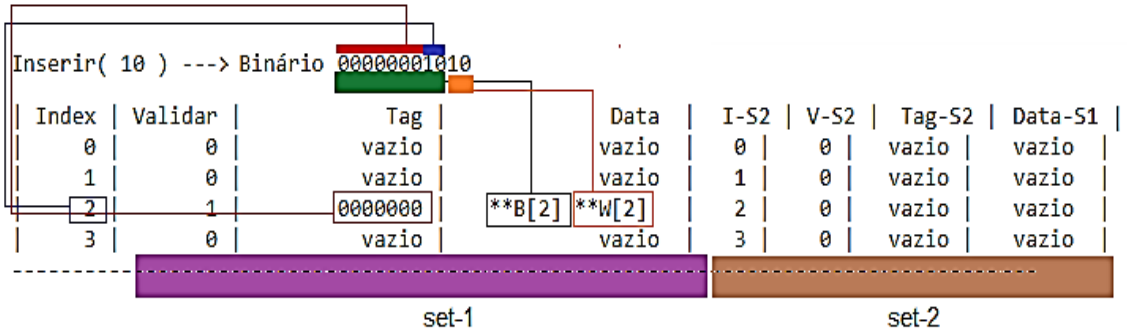

Figura 5 – Como funciona o Mapeamento Associativo por Conjunto no Simulador

Políticas de Substituição

Todas as políticas de substituição foram implementadas utilizando `arrays multidimensionais,` exceto a randômica. Neste caso, as políticas são definidas apenas para os tipos de mapeamento associativo por conjunto e totalmente associativo. Os próximos tópicos definem como cada uma dessas políticas foi realmente implementada no Python e seu funcionamento.

Random `(-sub Random)`**:** Esta é a estratégia mais simples de criar. Sendo assim, utilizamos a função Random do Python para definir de forma aleatória um intervalo numérico para substituição. Para este caso em particular, o intervalo definido é o tamanho total de índices que a memória cache possui. No caso do mapeamento por conjunto, a posição do conjunto também é determinada de forma randômica. Como temos apenas 2 conjuntos, a substituição é escolhida aleatoriamente entre o conjunto da direita e o conjunto da esquerda.

FIFO `(-sub FIFO)`**:** O algoritmo de FIFO é uma abreviação para *first in, first out*. Este método diz que o primeiro elemento processado é também o primeiro a ser removido. Neste caso, há um array para armazenamento de prioridade, onde o elemento mais ao topo na pilha de prioridade é o primeiro índice a ser substituído (ver Figura 6).

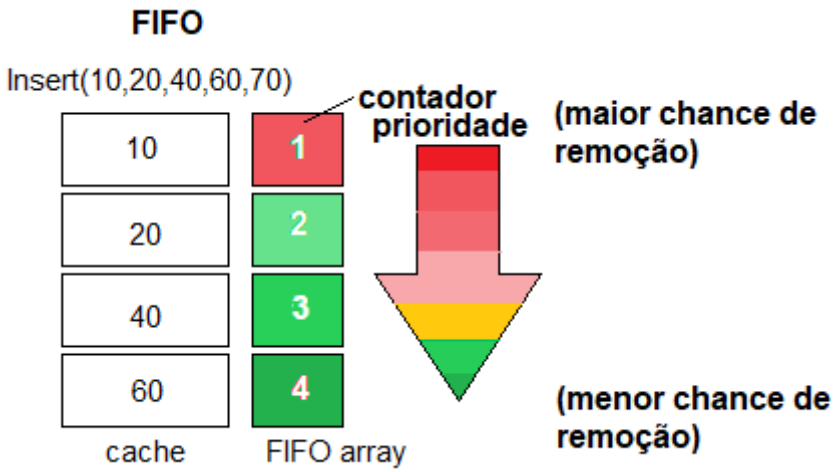

Figura 6 - Política de FIFO

LRU `(-sub LRU)`**:** O esquema de armazenamento em cache da LRU é para remover o quadro usado menos recentemente quando a memória cache está cheia e uma nova informação precisa ser colocada em um dos espaços. Para esta política, em ambos os casos de mapeamento por associação, há sempre um array com um contador, onde a cada acréscimo de uma informação este contador é incrementado. Com isso, o menor valor dentro do LRU array é a posição menos recentemente utilizada. Caso isso aconteça, ela é substituída pelo novo dado. Se ocorrer um empate, há um processo de randomização, escolhendo de forma randômica as posições que empataram (ver Figura 7).

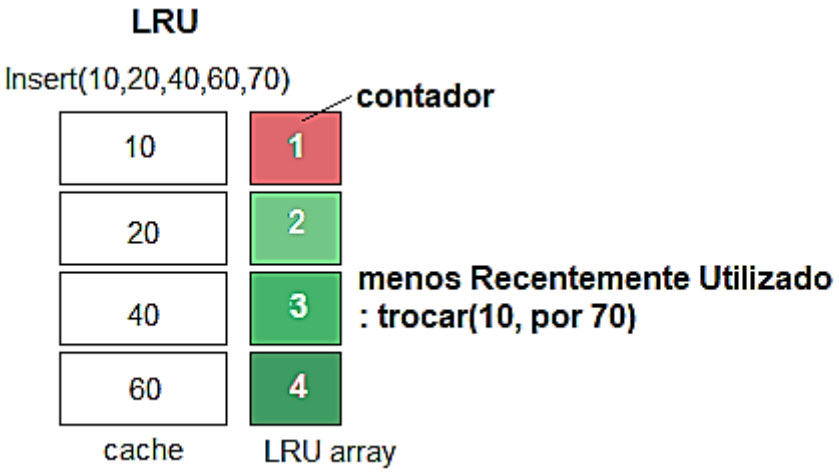

Figura 7 - Política LRU

LFU `(-sub LFU)`**:** No algoritmo de LFU é verificado a página antiga com menor frequência, e assim substitui-la por uma nova inserção. Neste caso, foi implementado um array de frequências, guardando todas as frequências de cada informação inserida na cache. Desta maneira, caso a memória cache esteja lotada, o índice com menor frequência é removido e substituído pelo novo dado naquele endereço (ver Figura 8).

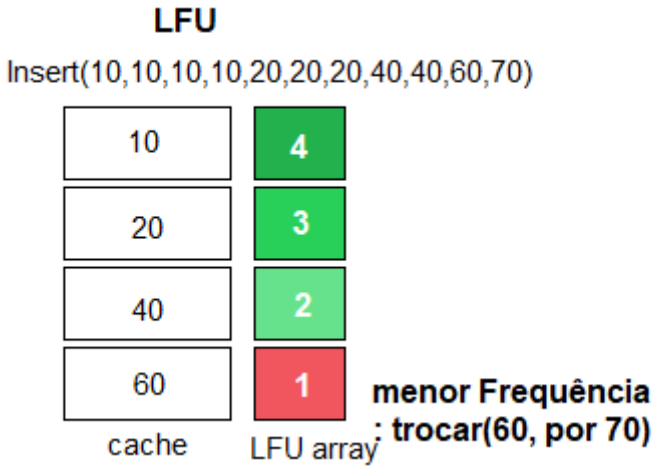

Figura 8 - Política de LFU

Comandos úteis (Atalhos)

- **Executar mapeamento direto com os valores no modo padrão do simulador (visualização completa):**

```
python run.py -map direto -local teste.txt -sub none -show yes
```

- **Executar mapeamento direto com os valores no modo padrão do simulador (visualização reduzida):**

```
python run.py -map direto -local teste.txt -sub none -show no
```

- **Executar mapeamento associativo por conjunto com os valores no modo padrão do simulador com a política de FIFO, LRU, LFU e Random:**

```
python run.py -map set-associative -local teste.txt -sub FIFO -show yes
python run.py -map set-associative -local teste.txt -sub LRU -show yes
python run.py -map set-associative -local teste.txt -sub LFU -show yes
python run.py -map set-associative -local teste.txt -sub Random -show yes
```

- **Executar mapeamento totalmente associativo com os valores no modo padrão do simulador com a política de FIFO, LRU, LFU e Random:**

```
python run.py -map fully-associative -local teste.txt -sub FIFO -show yes
python run.py -map fully-associative -local teste.txt -sub LRU -show yes
python run.py -map fully-associative -local teste.txt -sub LFU -show yes
python run.py -map fully-associative -local teste.txt -sub Random -show
yes
```

- **Executar automaticamente todos os tipos de mapeamento e cada uma das políticas definidas no simulador.**

```
python run.py -map all -sub all -local teste.txt
```

Anexo 1 – Interface de Conexão com o Script Principal

```
# coding: utf-8
from mapping import *;
import sys
import argparse
import os
import stat
import getpass

parser = argparse.ArgumentParser();
parser.add_argument("-show","--mostrarConteudoCache",help="Este parâmetro, se declarado como yes ou y, exibe todo conteúdo em todas as etapas de inserção na cache. Por padrão, está definido como 'yes'",default="yes");
parser.add_argument("-mp","--capacidadeMP",help="Capacidade Total da Memória Principal: Indique a Capacidade Total da Memória Principal, por padrão este valor é de 2048 bytes",default=4096);
parser.add_argument("-cache","--capacidadeCache",help="Capacidade Total da Memória Cache: Indique a Capacidade Total de Cache Disponível, por padrão este valor é de 16 bytes",default=32);
parser.add_argument("-w","--offsetBit",help="Tamanho da Palavra da Instrução em Bits: Indique a Capacidade de Bits para Representação do campo Palavras da Instrução, por padrão este valor é 2 (em bits)",default=2);
parser.add_argument("-map","--tipoDeMapeamento",help="Tipo de mapeamento: Indique o tipo de Mapeamento desejado. As opções válidas são: direto, fully-associative e set-associative. A representação set-associative por padrão é de 2 conjuntos apenas. Para os mapeamentos fully-associative e set-associative é necessário definir uma política de substituição",default='all');
parser.add_argument("-local","--diretorio",help="Diretorio ou caminho do arquivo: Indique o Caminho do Diretório Corresponde nte ao Arquivo .txt com os Dados de Entrada. Este arquivo deve estar com os valores em decimal, já que o script já realiza a conversão para binário.");
parser.add_argument("-sub","--politicaDeSubstituicao",help="Politica de Substituicao: Indique a politica de Substituicao desejada ou deixe este campo em branco para executar o programa como -all. O atributo -all realiza a execução do script com todas as políticas disponíveis (FIFO,LRU,LFU e Random)",default='all');
args = parser.parse_args();
mostrarAll = args.mostrarConteudoCache;
MemoriaPrincipal = int(args.capacidadeMP);
MemoriaCache = int(args.capacidadeCache);
PalavraBit = int(args.offsetBit);
Mapeamento = args.tipoDeMapeamento;
Politicas = args.politicaDeSubstituicao;
Caminho = r'{0}'.format(args.diretorio);
if(Mapeamento == 'all' and Politicas == 'all'):
    Executar = Memoria(MemoriaPrincipal, MemoriaCache, PalavraBit, 'direto', Caminho, 'none','no');
    SUB = ['FIFO','LRU','LFU','Random']
    for i in range(0,len(SUB)):
        Executar = Memoria(MemoriaPrincipal, MemoriaCache, PalavraBit, 'fully-associative', Caminho, SUB[i],'no');
    for i in range(0,len(SUB)):
        Executar = Memoria(MemoriaPrincipal, MemoriaCache, PalavraBit, 'set-associative', Caminho, SUB[i],'no');
    print("Modo de geração automática foi encerrado com sucesso!");
if(Mapeamento == "direto"):
    Executar = Memoria(MemoriaPrincipal, MemoriaCache, PalavraBit, Mapeamento, Caminho, "none", mostrarAll);
elif(Mapeamento == "fully-associative" or Mapeamento == "set-associative"):
    Executar = Memoria(MemoriaPrincipal, MemoriaCache, PalavraBit, Mapeamento, Caminho, Politicas, mostrarAll);
```

Anexo 2 – Interface do Simulador

```
1.  # coding: utf-8
2.
3.  # In[22]:
4.
5.
6.  import math;
7.  import random;
8.  import string
9.
10. class Memoria:
11.
12.     def __init__(self, capacidadeMP, capacidadeCache, offset, mapeamento, diretorio, escal
    onamento, show_all):
13.         # Cálculo e Desenvolvimento da Arquitetura da Memória Principal e Cache
14.         self.show_all = show_all;
15.         self.digs = string.digits + string.ascii_letters
16.         self.capacidadeMP = capacidadeMP;
17.         self.capacidadeCache = capacidadeCache;
18.         self.offset = offset;
19.         self.mapeamento = mapeamento;
20.         self.diretorio = diretorio;
21.         self.armazenarResposta = [];
22.         self.tamanhoBloco = int(math.pow(2,self.offset));
23.         # Construção da Estrutura da Memória Principal
24.         self.totalMPBloco = int(math.log2(self.capacidadeMP/self.tamanhoBloco));
25.         self.ultimoBloco = int(math.pow(2,self.totalMPBloco)-1);
26.         self.ultimaPalavra = int(math.pow(2,self.offset)-1);
27.         self.MPArrayBloco = ["**B["+ str(i)+"] " for i in range(self.ultimoBloco + 1)];
28.         self.MPArrayPalavra = ["**W["+ str(i)+"] " for i in range(self.ultimaPalavra + 1)]
    ;
29.         self.escalonamento = escalonamento;
30.
31.         if(self.mapeamento == "set-associative"):
32.             self.cacheIndex = int(math.log2((self.capacidadeCache/self.tamanhoBloco)/2));

33.             self.cacheTag = int(math.log2(self.capacidadeMP)) - self.offset - self.cacheIn
    dex;
34.             # Conjunto 1 - Lista de Arrays
35.             self.cacheIndexArray_1 = [i for i in range(int(math.pow(2, self.cacheIndex)))]
    ;
36.             self.cacheTagArray_1 = ["vazio" for i in range(int(math.pow(2, self.cacheIndex
    )))];
37.             self.cacheDataArray_1 = ["vazio" for i in range(int(math.pow(2, self.cacheInde
    x)))];
38.             self.cacheValidar_1 = [0 for i in range(int(math.pow(2, self.cacheIndex)))];
39.             # Conjunto 2 - Lista de Arrays
40.             self.cacheIndexArray_2 = [i for i in range(int(math.pow(2, self.cacheIndex)))]
    ;
41.             self.cacheTagArray_2 = ["vazio" for i in range(int(math.pow(2, self.cacheIndex
    )))];
42.             self.cacheDataArray_2 = ["vazio" for i in range(int(math.pow(2, self.cacheInde
    x)))];
43.             self.cacheValidar_2 = [0 for i in range(int(math.pow(2, self.cacheIndex)))];
44.             self.bitFIFO_2set = 0;
45.             # Conjunto de Pontuacoes para definir o candidato de substituição no LRU
46.             self.LRU_PontuacaoSET1 = [0 for i in range(int(math.pow(2, self.cacheIndex)))]
    ;
47.             self.LRU_PontuacaoSET2 = [0 for i in range(int(math.pow(2, self.cacheIndex)))]
    ;
48.             self.LRU_Contador = 0;
49.             self.controle = "";
50.             # Conjunto de Frequencias referentes aos endereços inseridos na cache no LFU
```

```
            self.LFU_Frequencia_S1 = [0 for i in range(int(math.pow(2, self.cacheIndex)))]
;
            self.LFU_Frequencia_S2 = [0 for i in range(int(math.pow(2, self.cacheIndex)))]
;
            self.LFU_contador = 0;
            self.LFU_CacheStatus_S1 = "vazia";
            self.LFU_Substituir = 0;
            self.LFU_Posicao = "Primeiro"
            # Inicialização e Leitura do Arquivo de Teste
            self.openFileExemplo(self.diretorio);
        elif(self.mapeamento == "fully-associative"):
            self.escalonamento = escalonamento;
            self.cacheIndex = int(math.log2((self.capacidadeCache/self.tamanhoBloco)));
            self.cacheTag = int(math.log2(self.capacidadeMP)) - self.offset;
            # Arrays para representar todos os Campos da Memória Cache
            self.cacheIndexArray = [i for i in range(int(math.pow(2, self.cacheIndex)))];

            self.cacheTagArray = ["vazio" for i in range(int(math.pow(2, self.cacheIndex))
)];
            self.cacheValidar = [0 for i in range(int(math.pow(2, self.cacheIndex)))];
            self.cacheDataArray = ["vazio" for i in range(int(math.pow(2, self.cacheIndex)
))];
            # Controle para determinar o candidato a ser substituido na política de FIFO;

            self.bitFIFO_fully = 0;
            # Controle para determinar o candidato a ser substituido na política de LRU
            self.LRU_Pontuacao = [0 for i in range(int(math.pow(2, self.cacheIndex)))];
            self.LRU_Contador = 0;
            self.inserirPosicao = 0;
            # Controle com o array de frequência para determinar o candidato a ser substit
uido na política de LFU
            self.LFU_Frequencia = [0 for i in range(int(math.pow(2, self.cacheIndex)))];
            self.LFU_contador = 0;
            self.LFU_CacheStatus = "vazia";
            self.LFU_Substituir = 0;
            # Inicialização e Leitura do Arquivo de Teste
            self.openFileExemplo(self.diretorio);
        elif(self.mapeamento == "direto"):
            # Criação dos arrays correspondentes ao Mapeamento Direto
            self.cacheIndex = int(math.log2((self.capacidadeCache/self.tamanhoBloco)));
            self.cacheTag = self.totalMPBloco - self.cacheIndex;
            self.cacheIndexArray = [i for i in range(int(math.pow(2, self.cacheIndex)))];

            self.cacheTagArray = ["vazio" for i in range(int(math.pow(2, self.cacheIndex))
)];
            self.cacheValidar = [0 for i in range(int(math.pow(2, self.cacheIndex)))];
            self.cacheDataArray = ["vazio" for i in range(int(math.pow(2, self.cacheIndex)
))];
            # Inicialização e Leitura do Arquivo de Teste
            self.openFileExemplo(self.diretorio);

        self.mostrarConfiguracao();
        self.fracaoDeAcertos();

    def int2base(self,x, base):
        if x < 0:
            sign = -1
        elif x == 0:
            return self.digs[0]
               else:
                   sign = 1
               x *= sign
               digits = []

               while x:
                   digits.append(self.digs[int(x % base)])
```

```
                    x = int(x / base)

                if sign < 0:
                    digits.append('-')
                digits.reverse()

                return ''.join(digits)

            def fracaoDeAcertos(self):
                # Cálculo da fração de acertos(cache hit e cache miss)
                contadorMiss = 0;
                for i in range(0,len(self.armazenarResposta)):
                    if(self.armazenarResposta[i] == "Miss"):
                        contadorMiss = contadorMiss + 1;
                # Função de cálculo em porcentagem
                fracaoMiss = "%.2f" % ((contadorMiss/len(self.armazenarResposta)) * 100);
                fracaoHit = "%.2f" % (100 - float(fracaoMiss));
                stringAcertos = 'Fração Miss:{0}%, Fração Hit:{1}%'.format(fracaoMiss,fraca
    oHit);
                # Mostra todos os acertos e erros de cada inserção
                print(stringAcertos);
                if(self.show_all == "yes" or self.show_all == "y"):
                    print(self.armazenarResposta);

                print("=" * 70);

            def politicasParaSetAssociative(self):
                idSubstituicao = 0;
                politica = self.escalonamento;
                # Política de Substituição Randomica
                if(politica == "Random"):
                    # Seleciona aleatóriamente uma posição para substituir do conjunto,ou s
    eja,
                    # Caso o valor seja igual a 0, o conjunto 1 é substituído com o valor r
    ecebido,
                    # Caso contrário, o conjunto 2 terá o elemento substituído na cache;
                    idSubstituicao = random.randint(0,1);
                # Política de Substituição FIFO
                elif(politica == "FIFO"):
                    # Seleciona com base na política de FIFO, como há apenas duas posições,
     este
                    # valor é somente alternado. Caso o acesso seja ao conjunto 1, o valor
    é 0
                    # Caso contrário, o conjunto 2 terá o elemento substituído na cache;
                    idSubstituicao = self.bitFIFO_2set;
                    if(self.bitFIFO_2set == 1):
                        self.bitFIFO_2set = 0;
                    else:
                        self.bitFIFO_2set = self.bitFIFO_2set + 1;
                # Política de Substituição LRU
                elif(politica == "LRU"):
                    # Caso a cache esteja cheia, o id da posição do vetor a ser substituído
     é o
                    # menor entre as pontuações, que são atualizadas na função a cada acess
    o na cache,
                    # seguindo as regras de LRU;
                    xMenor_conjunto1 = self.LRU_PontuacaoSET1.index(min(self.LRU_PontuacaoS
    ET1));
                    xMenor_conjunto2 = self.LRU_PontuacaoSET2.index(min(self.LRU_PontuacaoS
    ET2));
                    if(xMenor_conjunto1 < xMenor_conjunto2):
                        self.LRU_PontuacaoSET1[xMenor_conjunto1] = self.LRU_Contador + 1;
                        idSubstituicao = xMenor_conjunto1;
                        self.controle = "Primeiro"
                    elif(xMenor_conjunto2 < xMenor_conjunto1):
```

```
163.                         # Caso aconteça um empate, o valor é escolhido de forma aleatória, onde
164.                         # o PRIMEIRO, corresponde ao conjunto 1 e o SEGUNDO ao conjunto 2.
165.                         # Este valor se alterna para evitar que uma mesma posição seja retirada sempre,
166.                         # já que apenas dois conjuntos estão representados pelo set-associative mapping neste caso.
167.                         self.LRU_PontuacaoSET2[xMenor_conjunto2] = self.LRU_Contador + 1;
168.                         idSubstituicao = xMenor_conjunto2;
169.                         self.controle = "Segundo";
170.                     else:
171.                         self.LRU_PontuacaoSET2[xMenor_conjunto2] = self.LRU_Contador + 1;
172.                         idSubstituicao = xMenor_conjunto2;
173.                         self.controle = "Segundo";
174.
175.                 # Política de Substituição de LFU
176.                 elif(politica == "LFU"):
177.                     # Semelhante ao funcionamento da função anterior (LRU)
178.                     # Porém, aqui o trabalho é com frequências de acesso!
179.                     xMenor_conjunto1 = self.LFU_Frequencia_S1.index(min(self.LFU_Frequencia_S1));
180.                     xMenor_conjunto2 = self.LFU_Frequencia_S2.index(min(self.LFU_Frequencia_S2));
181.                     if(xMenor_conjunto1 < xMenor_conjunto2):
182.                         self.LFU_Frequencia_S1[xMenor_conjunto1] = 1;
183.                         idSubstituicao = xMenor_conjunto1;
184.                         self.LFU_Posicao = "Primeiro"
185.                         # Mesma ideia implementada para o LRU em caso de empate
186.                     elif(xMenor_conjunto2 < xMenor_conjunto1):
187.                         self.LFU_Frequencia_S2[xMenor_conjunto2] = 1;
188.                         idSubstituicao = xMenor_conjunto2;
189.                         self.LFU_Posicao = "Segundo";
190.                     elif(xMenor_conjunto1 == xMenor_conjunto2):
191.                         randomValue = random.randint(0,1);
192.                         if(randomValue == 0):
193.                             self.LFU_Frequencia_S1[xMenor_conjunto1] = 1; idSubstituicao = xMenor_conjunto1; self.LFU_Posicao = "Primeiro";
194.                         else:
195.                             self.LFU_Frequencia_S2[xMenor_conjunto2] = 1; idSubstituicao = xMenor_conjunto2; self.LFU_Posicao = "Segundo";
196.
197.                 # Retornar o valor de substituição obtido com uma das políticas dessa função;
198.                 return idSubstituicao;
199.
200.
201.             def politicaDeSubstituicao(self):
202.                 idSubstituicao = 0;
203.                 # Chama a função de política para o associativo por conjunto.
204.                 # Já que o mapeamento direto não possui política de substituição
205.                 # e o mapeamento totalmente associativo possui as políticas implementadas
206.                 # dentro da própria função
207.                 if(self.mapeamento == "set-associative"):
208.                     idSubstituicao = self.politicasParaSetAssociative();
209.                 else:
210.                     print("Não há politicas de substituição para o Mapeamento Direto");
211.                 # Retornar o id de Substituição, isto é, a posição do vetor;
212.                 return idSubstituicao;
213.
214.             def mapeamentoDireto(self,i,t,o):
215.                 # Mapeamento Direto
216.                 indexDecimal = int(i,2); # Converter para decimal
217.                 palavraDecimal = int(o,2); # Converter para decimal
218.                 # Inserir dado com base no endereço de index
219.                 if(self.cacheValidar[indexDecimal] == 0):
```

```
                    self.cacheTagArray[indexDecimal] = t;
                    self.cacheDataArray[indexDecimal] = self.MPArrayBloco[int(t + i,2)] + self.MPArrayPalavra[palavraDecimal];
                    self.cacheValidar[indexDecimal] = 1;
                    self.armazenarResposta.append("Miss");
                # Verificar se determinada posição(index) na cache está preenchido
                elif(self.cacheValidar[indexDecimal] == 1):
                    # Operação em caso de cache hit
                    if(self.cacheTagArray[indexDecimal] == t):
                        self.armazenarResposta.append("Hit");
                    # Operação em caso de cache miss
                    else:
                        self.cacheTagArray[indexDecimal] = t;
                        self.cacheDataArray[indexDecimal] = self.MPArrayBloco[int(t + i,2)] + self.MPArrayPalavra[palavraDecimal];
                        self.cacheValidar[indexDecimal] = 1;
                        self.armazenarResposta.append("Miss");

            def mapeamentoFullyAssociative(self,t,o):
                # Mapeamento Totalmente Associativo, esta função recebe as variaveis
                # correspondentes ao campo tag e offset-bit, visto que não há index
                politica = self.escalonamento;
                resposta = "";
                palavraDecimal = int(o,2); # Converter para Binário
                # Em caso da política escolhida for a FIFO(First In, First Out)
                if(politica == "FIFO"):
                    # Busca em todas as posições da cache, visto que não há index
                    for i in range(0,len(self.cacheIndexArray)):
                        # Percorrer e procurar um elemento com a mesma tag (cache hit)
                        if(self.cacheTagArray[i] == t):
                            self.armazenarResposta.append("Hit");
                            resposta = "Sim";
                        # Caso contrário, inserir em qualquer posição, baseando-se na
                        # política de substituição escolhida
                    if(resposta != "Sim"):
                        # No FIFO, a substituição ou inserção se baseia no id do array inserido primeiro.
                        # Esta função está baseada na ordem de acesso aos endereços solicitados.
                        self.cacheTagArray[self.bitFIFO_fully] = t;
                        self.cacheDataArray[self.bitFIFO_fully] = self.MPArrayBloco[int(t,2)] + self.MPArrayPalavra[palavraDecimal];
                        self.cacheValidar[self.bitFIFO_fully] = 1;
                        self.armazenarResposta.append("Miss");
                        if(self.bitFIFO_fully == len(self.cacheIndexArray) - 1):
                            self.bitFIFO_fully = 0;
                        else:
                            self.bitFIFO_fully = self.bitFIFO_fully + 1;
                # Política de Substituição LFU (Por frequência de acessos, menor frequência é eliminado primeiro)
                elif(politica == "LFU"):
                    # Percorrer e procurar um elemento com a mesma tag (cache hit)
                    for i in range(0,len(self.cacheIndexArray)):
                        if(self.cacheTagArray[i] == t):
                            self.armazenarResposta.append("Hit");
                            resposta = "Sim";
                            # Há um vetor de frequências, que armazena as frequências de cada
                            # um dos elementos, baseando-
                    se na sua posição dentro do array.
                            self.LFU_Frequencia[i] = self.LFU_Frequencia[i] + 1;
                    # Caso contrário, inserir em qualquer posição, baseando-se na
                    # política de substituição escolhida
                    if(resposta != "Sim"):
                        # Esta função é responsável por retornar o id do array com menor frequência.
```

```
277.                         # Em caso de empate, o elemento mais perto do topo do array é selec
    ionado, ou seja
278.                         # o mais próximo da posição LFU_Frequencia[0]
279.                         menorFrequencia = self.LFU_Frequencia.index(min(self.LFU_Frequencia
    ));
280.                         self.LFU_Frequencia[menorFrequencia] = 1;
281.                         self.cacheTagArray[menorFrequencia] = t;
282.                         self.cacheDataArray[menorFrequencia] = self.MPArrayBloco[int(t,2)]
    + self.MPArrayPalavra[palavraDecimal];
283.                         self.cacheValidar[menorFrequencia] = 1;
284.                         self.armazenarResposta.append("Miss");
285.                 # Em caso da política escolhida for a Random
286.                 elif(politica == "Random"):
287.                     # Percorrer e procurar um elemento com a mesma tag (cache hit)
288.                     for i in range(0,len(self.cacheIndexArray)):
289.                         if(self.cacheTagArray[i] == t):
290.                             self.armazenarResposta.append("Hit");
291.                             resposta = "Sim";
292.                             # Há um vetor de frequências, que armazena as frequências de ca
    da
293.                             # um dos elementos, baseando-
    se na sua posição dentro do array.
294.                     # política de substituição escolhida
295.                     if(resposta != "Sim"):
296.                         # Esta função gera randomicamente uma posição de array entre as sel
    ecionadas.
297.                         # Os valores gerados são totalmente aleatórios, logo não é possível
     determinar
298.                         # em qual posição o dado será inserido ou substituido
299.                         x = random.randint(0,(len(self.cacheIndexArray) - 1));
300.                         self.cacheTagArray[x] = t;
301.                         self.cacheDataArray[x] = self.MPArrayBloco[int(t,2)] + self.MPArray
    Palavra[palavraDecimal];
302.                         self.cacheValidar[x] = 1;
303.                         self.armazenarResposta.append("Miss");
304.                         if(self.bitFIFO_fully == len(self.cacheIndexArray) - 1):
305.                             self.bitFIFO_fully = 0;
306.                         else:
307.                             self.bitFIFO_fully = self.bitFIFO_fully + 1;
308.                 # Política de Substituição LRU(Recentemente utilizado)
309.                 elif(politica == "LRU"):
310.                     # O elemento menos recentemente utilizado é eliminado em caso de confli
    to
311.                     # Percorrer e procurar um elemento com a mesma tag (cache hit)
312.                     for i in range(0,len(self.cacheIndexArray)):
313.                         if(self.cacheTagArray[i] == t):
314.                             self.armazenarResposta.append("Hit");
315.                             resposta = "Sim";
316.                             # Há um vetor de frequências, que armazena as frequências de ca
    da
317.                             # um dos elementos, baseando-
    se na sua posição dentro do array.
318.                     # Política de Substituição selecionada
319.                     if(resposta != "Sim"):
320.                         # Há um contador e um vetor de Pontuações com base na posição do ve
    tor de cada elemento.
321.                         # Assim, a substituição ocorre com o elemento de menor pontuação (m
    enos recentemente utilizado)
322.                         self.cacheTagArray[self.inserirPosicao] = t;
323.                         self.LRU_Pontuacao[self.inserirPosicao] = self.LRU_Contador;
324.                         self.LRU_Contador = self.LRU_Contador + 1;
325.                         self.cacheDataArray[self.inserirPosicao] = self.MPArrayBloco[int(t,
    2)] + self.MPArrayPalavra[palavraDecimal];
326.                         self.cacheValidar[self.inserirPosicao] = 1;
327.                         self.armazenarResposta.append("Miss");
328.                         # Em caso da cache estar cheia !
```

```
329.                         k = 0;
330.                         for i in range(len(self.cacheIndexArray)):
331.                             if(self.cacheValidar[i] == 1):
332.                                 k = k + 1;
333.                         # Cálcular o menor elemento entre as pontuações no vetor LRU_Pontua
    ção
334.                         if(k == len(self.cacheIndexArray)):
335.                             xMenor = self.LRU_Pontuacao.index(min(self.LRU_Pontuacao))
336.                             self.inserirPosicao = xMenor;
337.                             self.LRU_Pontuacao[xMenor] = self.LRU_Pontuacao[xMenor] + 1;
338.                             # O menor elemento é selecionado e somado, para indicar que est
    e teve um acesso.
339.                             # A posição é armazenada e alterada na próxima rodada do algori
    tmo.
340.                         else:
341.                             self.inserirPosicao = self.inserirPosicao + 1;
342.
343.
344.             def mapeamentoSetAssociative(self,i,t,o):
345.                 # Mapeamento Associativo por Conjunto, definido com um tamanho fixo de 2 co
    njuntos no total;
346.                 # Neste, as operações de controle são mais complexas, já que é necessário a
    dministrar duas
347.                 # tabelas diferentes (arrays), com base na política de substituição selecio
    nada
348.                 indexDecimal = int(i,2);
349.                 palavraDecimal = int(o,2);
350.                 if(self.cacheValidar_1[indexDecimal] == 0) and (self.cacheValidar_2[indexDe
    cimal] == 0):
351.                     self.cacheTagArray_1[indexDecimal] = t;
352.                     self.cacheValidar_1[indexDecimal] = 1;
353.                     self.cacheDataArray_1[indexDecimal] = self.MPArrayBloco[int(t + i,2)] +
     self.MPArrayPalavra[palavraDecimal]
354.                     self.armazenarResposta.append("Miss");
355.                     # Armazena a potuação do LRU e também armazena a frequência do LFU.
356.                     # Como estamos percorrendo os vetores, esta função foi aproveitada para

357.                     # realizar todos os cálculos ao mesmo tempo, da LRU e LFU;
358.                     self.LRU_PontuacaoSET1[indexDecimal] = self.LRU_Contador + 1; # Contado
    r para o conjunto 1
359.                     self.LFU_Frequencia_S1[indexDecimal] = self.LFU_Frequencia_S1[indexDeci
    mal] + 1; # Frequencia para o conjunto 1
360.                 # Próximo passo, verificar se um dos conjuntos possue espaço.
361.                 elif(self.cacheValidar_1[indexDecimal] == 1) and (self.cacheValidar_2[index
    Decimal] == 0):
362.                     if(self.cacheTagArray_1[indexDecimal] == t):
363.                         self.armazenarResposta.append("Hit");
364.                         # Array de Pontuação do LRU e frequência do LFU - Conjunto 1;
365.                         self.LRU_PontuacaoSET1[indexDecimal] = self.LRU_Contador + 1;
366.                         self.LFU_Frequencia_S1[indexDecimal] = self.LFU_Frequencia_S1[index
    Decimal] + 1;
367.                     else:
368.                         self.cacheTagArray_2[indexDecimal] = t;
369.                         self.cacheValidar_2[indexDecimal] = 1;
370.                         self.cacheDataArray_2[indexDecimal] = self.MPArrayBloco[int(t + i,2
    )] + self.MPArrayPalavra[palavraDecimal]
371.                         self.armazenarResposta.append("Miss");
372.                         # Array de Pontuação do LRU e frequência do LFU - Conjunto 2;
373.                         self.LRU_PontuacaoSET2[indexDecimal] = self.LRU_Contador + 1;
374.                         self.LFU_Frequencia_S2[indexDecimal] = self.LFU_Frequencia_S2[index
    Decimal] + 1;
375.                 # Caso a cache esteja cheia em ambos os conjuntos:
376.                 elif(self.cacheValidar_1[indexDecimal] == 0) and (self.cacheValidar_2[index
    Decimal] == 1):
377.                     if(self.cacheTagArray_2[indexDecimal] == t):
378.                         self.armazenarResposta.append("Hit");
```

```
                        # Array de Pontuação do LRU e frequência do LFU - Conjunto 2, em caso de HIT;
                        self.LRU_PontuacaoSET2[indexDecimal] = self.LRU_Contador + 1;
                        self.LFU_Frequencia_S2[indexDecimal] = self.LFU_Frequencia_S2[indexDecimal] + 1;
                    # Caso seja um cache miss:
                    else:
                        self.cacheTagArray_1[indexDecimal] = t;
                        self.cacheValidar_1[indexDecimal] = 1;
                        self.cacheDataArray_1[indexDecimal] = self.MPArrayBloco[int(t + i,2)] + self.MPArrayPalavra[palavraDecimal]
                        self.armazenarResposta.append("Miss");
                         # Array de Pontuação do LRU e frequência do LFU - Conjunto 1, em caso de MISS;
                        self.LRU_PontuacaoSET1[indexDecimal] = self.LRU_Contador + 1;
                        self.LFU_Frequencia_S1[indexDecimal] = self.LFU_Frequencia_S1[indexDecimal] + 1;
                # Caso ambos os espaços esteja ocupados - bit validator = 1 em ambos os conjuntos
                elif(self.cacheValidar_1[indexDecimal] == 1) and (self.cacheValidar_2[indexDecimal] == 1):
                        if(self.cacheTagArray_1[indexDecimal] == t):
                            self.armazenarResposta.append("Hit");
                             # Operação em caso de um hit no conjunto 1
                            self.LRU_PontuacaoSET1[indexDecimal] = self.LRU_Contador + 1;
                            self.LFU_Frequencia_S1[indexDecimal] = self.LFU_Frequencia_S1[indexDecimal] + 1;
                        elif(self.cacheTagArray_2[indexDecimal] == t):
                            self.armazenarResposta.append("Hit");
                             # Operação em caso de um hit no conjunto 2
                            self.LRU_PontuacaoSET2[indexDecimal] = self.LRU_Contador + 1;
                            self.LFU_Frequencia_S2[indexDecimal] = self.LFU_Frequencia_S2[indexDecimal] + 1;
                        # Caso ocorra um MISS:
                        else:
                            # Chama a função que cálcula a posição do vetor cache que será substituido,
                            # conforme as políticas de substituições adotados e selecionadas;
                            escolherSubstituicao = self.politicaDeSubstituicao();
                            if(self.escalonamento == "LFU"):
                                # Em caso da política ser LFU:
                                if(self.LFU_Posicao == "Primeiro"):
                                    # Realizar a substituição no primeiro conjunto (1)
                                    self.armazenarResposta.append("Miss");
                                    self.cacheTagArray_1[indexDecimal] = t;
                                    self.cacheValidar_1[indexDecimal] = 1;
                                    self.cacheDataArray_1[indexDecimal] = self.MPArrayBloco[int(t + i,2)] + self.MPArrayPalavra[palavraDecimal]
                                elif(self.LFU_Posicao == "Segundo"):
                                    # Realizar a substituição no segundo conjunto (2)
                                    self.armazenarResposta.append("Miss");
                                    self.cacheTagArray_1[indexDecimal] = t;
                                    self.cacheValidar_1[indexDecimal] = 1;
                                    self.cacheDataArray_1[indexDecimal] = self.MPArrayBloco[int(t + i,2)] + self.MPArrayPalavra[palavraDecimal]
                            # Em caso da política ser LRU
                            elif(self.escalonamento == "LRU"):
                                if(self.controle == "Primeiro"):
                                    # Realizar a substituição no primeiro conjunto (1)
                                    self.armazenarResposta.append("Miss");
                                    self.cacheTagArray_1[indexDecimal] = t;
                                    self.cacheValidar_1[indexDecimal] = 1;
                                    self.cacheDataArray_1[indexDecimal] = self.MPArrayBloco[int(t + i,2)] + self.MPArrayPalavra[palavraDecimal]
                                elif(self.controle == "Segundo"):
```

```
                                    # Realizar a substituição no segundo conjunto (2)
                                    self.armazenarResposta.append("Miss");
                                    self.cacheTagArray_1[indexDecimal] = t;
                                    self.cacheValidar_1[indexDecimal] = 1;
                                    self.cacheDataArray_1[indexDecimal] = self.MPArrayBloco
    [int(t + i,2)] + self.MPArrayPalavra[palavraDecimal]
                            # Em caso da política ser Random ou FIFO.
                            # Esta estrutura é a mesma para ambas as políticas, para aprove
    itamento do
                            # código, já que é apenas o conteúdo de uma determinada posição
     que é trocado.
                            # A função responsável por fazer a troca com base na política s
    elecionada, é a função politicaDeSubstituicao()
                            else:
                                if(escolherSubstituicao == 0):
                                    self.armazenarResposta.append("Miss");
                                    self.cacheTagArray_1[indexDecimal] = t;
                                    self.cacheValidar_1[indexDecimal] = 1;
                                    self.cacheDataArray_1[indexDecimal] = self.MPArrayBloco
    [int(t + i,2)] + self.MPArrayPalavra[palavraDecimal]
                                else:
                                    self.armazenarResposta.append("Miss");
                                    self.cacheTagArray_2[indexDecimal] = t;
                                    self.cacheValidar_2[indexDecimal] = 1;
                                    self.cacheDataArray_2[indexDecimal] = self.MPArrayBloco
    [int(t + i,2)] + self.MPArrayPalavra[palavraDecimal]

                            self.LRU_Contador = self.LRU_Contador + 1;

            def openFileExemplo(self, diretorio):
                # Calcular o Tamanho Total da Instrução recebida
                tamanhoDaInstrucao = int(math.log2(self.capacidadeMP));
                # Abrir o arquivo conforme o diretório informado como parâmetro
                arquivo = open(diretorio, "r");
                items = arquivo.readlines();
                array_values = [];
                for i in range(len(items)):
                    v = int(items[i].replace("\n",""));
                    array_values.append(v)

                for i in range(0,len(array_values)):
                    Decimal = str(array_values[i])
                    DecimalToBinario = self.int2base(array_values[i],2)
                    # Acrescentar zeros nos bits, conforme o tamanho da instrução
                    bitZero = tamanhoDaInstrucao - len(DecimalToBinario);
                    linha = "0" * bitZero + DecimalToBinario;
                    # Formatar valores, conforme as linhas recebidas (em binário)
                    i, t, o = self.formatar(linha);
                    # Informa a inserção atual, o programa mostra o conteúdo da
                    # memória cache a cada rodada.
                    if(self.show_all == "y" or self.show_all == "yes"):
                        print("Inserir(",Decimal,") ---> Binário:",linha,"\n");

                    if(self.mapeamento == "set-associative"):
                        # Chama a função de set-associative mapping
                        self.mapeamentoSetAssociative(i,t,o);
                    elif(self.mapeamento == "fully-associative"):
                        # Chama a função de fully-associative mapping
                        self.mapeamentoFullyAssociative(t,o);
                    else:
                            # Chama a função de direct-associative mapping
                        self.mapeamentoDireto(i,t,o);
                        # Mostra o conteúdo do vetor (memória cache) a cada rodada;

                    if(self.show_all == "y" or self.show_all == "yes"):
                        self.mostrarConteudo();
```

```
491.                     print("-" * 100);
492.
493.         def formatar(self, enderecoExemplo):
494.             # Formata a entrada conforme os campos solicitados por cada estrutura de ma
    peamento
495.             if(self.mapeamento == "set-associative"):
496.                 tagFinal = self.cacheTag - 1;
497.                 cacheIndexInicial = tagFinal + 1;
498.                 cacheIndexFinal = cacheIndexInicial + (self.cacheIndex - 1);
499.                 offsetInicial = cacheIndexFinal + 1;
500.                 offsetFinal = offsetInicial + (self.offset - 1);
501.                 tagExemplo = enderecoExemplo[0:tagFinal + 1];
502.                 indexExemplo = enderecoExemplo[cacheIndexInicial:cacheIndexFinal + 1];

503.                 offsetExemplo = enderecoExemplo[offsetInicial:offsetFinal + 1];
504.             elif(self.mapeamento == "fully-associative"):
505.                 tagFinal = self.cacheTag - 1;
506.                 offsetInicial = tagFinal + 1;
507.                 offsetFinal = offsetInicial + (self.offset - 1);
508.                 tagExemplo = enderecoExemplo[0:tagFinal + 1];
509.                 offsetExemplo = enderecoExemplo[offsetInicial:offsetFinal + 1];
510.                 indexExemplo = 0;
511.             else:
512.                 tagFinal = self.cacheTag - 1;
513.                 cacheIndexInicial = tagFinal + 1;
514.                 cacheIndexFinal = cacheIndexInicial + (self.cacheIndex - 1);
515.                 offsetInicial = cacheIndexFinal + 1;
516.                 offsetFinal = offsetInicial + (self.offset - 1);
517.                 tagExemplo = enderecoExemplo[0:tagFinal + 1];
518.                 indexExemplo = enderecoExemplo[cacheIndexInicial:cacheIndexFinal + 1];

519.                 offsetExemplo = enderecoExemplo[offsetInicial:offsetFinal + 1];
520.             # Retorna os valores de Index (caso exista), Tag e offSet-bit
521.             return indexExemplo, tagExemplo, offsetExemplo;
522.
523.         def mostrarConteudo(self):
524.             # Mostrar conteúdo com base na formatação do script:
525.             if(self.mapeamento == "set-associative"):
526.                 stringTable = '| {i1:>3} | {v1:>4} | {t1:>9} | {d1:>20}  || {i2:>3} | {
    v2:>4} | {t2:>15} | {d2:>20}  |'.format(
527.                         i1 = 'I-S1',v1 = 'V-S1',t1 = 'Tag-S1', d1 = 'Data-S1',
528.                         i2 = 'I-S2',v2 = 'V-S2',t2 = 'Tag-S2', d2 = 'Data-S1');
529.                 print(stringTable);
530.                 for i in range(int(math.pow(2, self.cacheIndex))):
531.                     stringCache = '| {i1:>3} | {v1:>4} | {t1:>9} | {d1:>20}  || {i2:>3}
     | {v2:>4} | {t2:>15} | {d2:>20}  |'.format(
532.                                                                         i1 = self.cacheI
    ndexArray_1[i],v1 = self.cacheValidar_1[i],
533.                                                                         t1 = self.cacheT
    agArray_1[i], d1 = self.cacheDataArray_1[i],
534.                                                                         i2 = self.cacheI
    ndexArray_2[i],v2 = self.cacheValidar_2[i],
535.                                                                         t2 = self.cacheT
    agArray_2[i], d2 = self.cacheDataArray_2[i],);
536.                     print(stringCache);
537.
538.             elif(self.mapeamento == "fully-
    associative" or self.mapeamento == "direto"):
539.                 stringTable = '| {i:>5} | {v:>7} | {t:>15} | {d:>20}  |'.format(i = 'In
    dex',v = 'Validar',t = 'Tag', d = 'Data');
540.                 print(stringTable);
541.                 for i in range(int(math.pow(2, self.cacheIndex))):
542.                     stringCache = '| {i:>5} | {v:>7} | {t:>15} | {d:>20}  |'.format(i =
     self.cacheIndexArray[i],
543.                                                                                 v =
     self.cacheValidar[i],
```

```
                                                                                t =
    self.cacheTagArray[i],
                                                                                d =
    self.cacheDataArray[i]);
                        print(stringCache);
                else:
                    print("Erro ao Exibir, mapeamento inserido não corresponde a nenhum dos
    listados!");

            def mostrarConfiguracao(self):
                if(self.mapeamento == "direto"):
                    self.escalonamento = "nenhum"
                # Exibir todas as configurações e cálculos realizados
                # Isso inclui os valores de index, tag, tipo de mapeamento e tipo de políti
    ca de Substituição
                print("=" * 70);
                stringConfiguracao = 'Capacidade MB:{0} (em bytes), nBlocos(K):{1}, Capacid
    ade cache:{2} (em bytes)'.format(self.capacidadeMP,self.tamanhoBloco,

                 self.capacidadeCache);
                stringMP = 'MP possui blocos de B[0..{0}] com palavras de W[0..{1}]'.format
    (self.ultimoBloco,self.ultimaPalavra);
                print(stringConfiguracao,'\n',stringMP);
                if(self.mapeamento == "direto" or self.mapeamento == "set-associative"):
                    stringCache = '--
    > Mapeamento:{0}   [tag:{1}, index:{2}, offset:{3}]'.format(self.mapeamento,self.cacheTag,
    self.cacheIndex,self.offset);
                    print(stringCache,'\n Politica de Substituição:',self.escalonamento);
                else:
                    stringCache = '--
    > Mapeamento:{0}   [tag:{1}, offset:{2}]'.format(self.mapeamento,self.cacheTag,self.offset
    );
                    print(stringCache,'\n Politica de Substituição:',self.escalonamento);
                # Fim
                print("=" * 70);
```