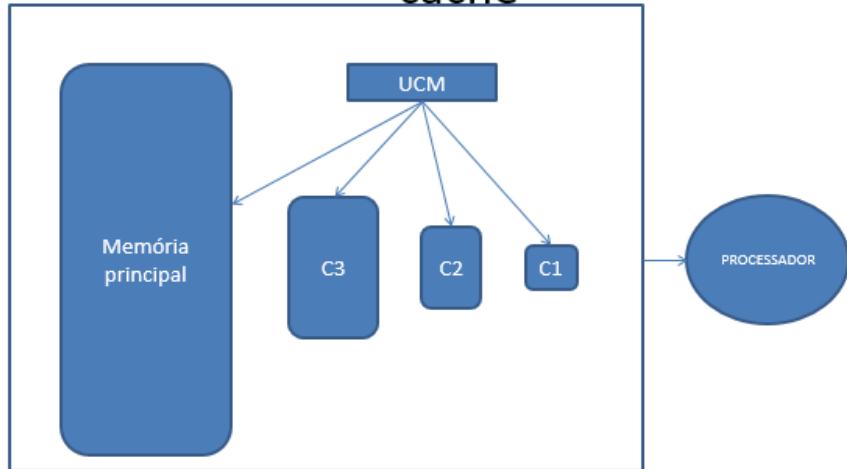


## Trabalho Prático da Disciplina BCC 266 - TP 2

Neste trabalho, o aluno entrará em contato com sistemas de memória, particularmente com o sistema cache.

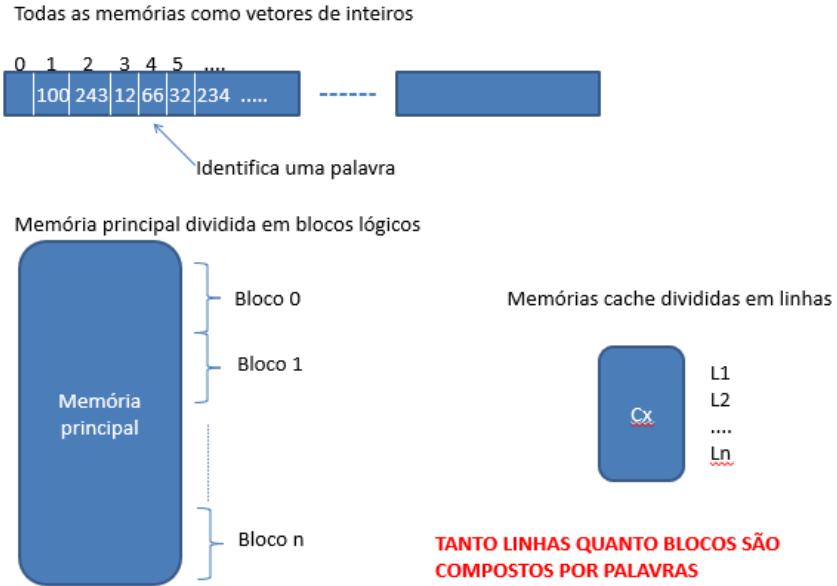
O aluno irá adicionar um sistema de cache em 3 níveis ao TP1, assim como ilustra a figura a seguir.

### Simulando o sistema de memória cache



Memória principal > C3 > C2 > C1

Assim como no TP1, as memórias possuem conteúdos denominados palavras e estas podem ser do tipo inteiro, booleano, byte ou qualquer outro suportado pela linguagem escolhida para fazer o TP. A memória principal é partitionada em blocos e a memória cache é partitionada em linhas, conforme ilustra a figura a seguir:



Simule o mapeamento associativo ou o mapeamento associativo em conjunto para a troca de linhas (e consequentemente palavras) entre as caches e a memória principal. Não pode ser o mapeamento direto disponibilizado pelo professor no site da disciplina. Este serve apenas como base para a construção dos demais.

Para substituição de linhas de cache, use alguma das políticas explicadas pelo professor, sejam elas LRU ou LFU. Para política de escrita, especifique se usou write-through ou write-back.

Por fim, utilize o gerador de instruções, disponível no site da disciplina para as linguagens Java, C++, Python, JavaScript e outras. Tais geradores simulam as repetições de instruções necessárias para o efetivo resultado das memórias cache.

Resultados: Na forma de tabelas (veja a figura a seguir), ilustrando cache hit e cache miss, assim como tempo hipotético de execução do programa. Faça isto para TODOS os tipos de máquinas testadas. Altere os tamanhos de cache, o número de caches , o nível de repetição de instruções e as políticas de substituição.

## RESULTADOS de forma TABULAR

50% de repetição, 75% de repetição, 90% de repetição

	Cache 1	Cache 2	Cache 3	Taxa C1 %	Taxa C2 %	Taxa C3 %	Taxa de RAM %	Taxa de disco %	Tempo de Execução (unidade hipotética)
M1	8	16	32						
M2	32	64	128						
M3	16	64	256						
M4	8	32	128						
M5	16	32	64						

Por fim, veja como vai ser a ideia de estender o TP1 para construir o TP2. Inicialmente, o aluno vai ter que criar as memórias cache níveis 1, 2 e 3, similar à memória RAM. Uma diferença substancial é que as memórias não serão mais vetores de inteiros simples, mas sim vetores de blocos no caso da RAM e vetores de linhas no caso das memórias cache. Cada bloco ou cada linha é um vetor de inteiros com 4 palavras.

Veja como ficaram as memórias de uma forma bem simplificada. Neste caso, as memórias caches são feitas também por blocos, algo que não corresponde à realidade, mas que iremos aceitar no TP2 por questões de simplificação:

```

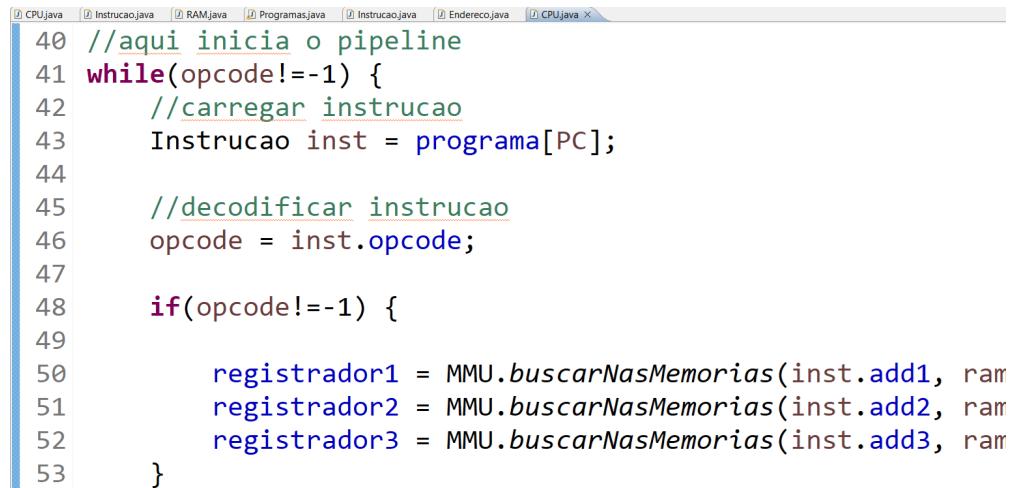
CPU.java  Instrucao.java  RAM.java  Programas.java  Instrucao.java  Endereco.java  CPU.java X
1 package bcc266TP2.toy;
2
3 public class CPU {
4     //a cpu pode ter dezenas de registradores
5     BlocoMemoria registrador1;
6     BlocoMemoria registrador2;
7     BlocoMemoria registrador3;
8
9     BlocoMemoria[] cache1;
10    BlocoMemoria[] cache2;
11
12    Instrucao[] programa;
13}

```

Veja como é fácil a estrutura de um bloco:

```
public class BlocoMemoria {  
  
    //BlocoMemoria.palavras[0] => um dado da ram  
  
    int[] palavras;  
    int endBloco;  
    boolean atualizado;  
    int custo;  
    int cacheHit;
```

Na máquina interpretada, o aluno terá que mudar o acesso à RAM, pois agora há também o acesso às memórias caches. Iremos encapsular todo o acesso ao sistema de memória numa função chamada UCM (Unidade de Controle de Memória). Veja a seguir onde foram feitas as intervenções na máquina interpretada do TP1 para adicionarmos as chamadas à função do TP2 denominada UCM.



```
40 //aqui inicia o pipeline  
41 while(opcode!=-1) {  
42     //carregar instrucao  
43     Instrucao inst = programa[PC];  
44  
45     //decodificar instrucao  
46     opcode = inst.opcode;  
47  
48     if(opcode!=-1) {  
49  
50         registrador1 = MMU.buscarNasMemorias(inst.add1, ram);  
51         registrador2 = MMU.buscarNasMemorias(inst.add2, ram);  
52         registrador3 = MMU.buscarNasMemorias(inst.add3, ram);  
53     }  
}
```