**Instituto Federal de Ciência e Tecnologia do Ceará**

Simulador de Memória Cache
Relatório

**Curso:** Ciência da Computação
**Aluno:** Samuel Mendes Rodrigues

## 1. Introdução

O objetivo desse relatório é analisar os resultados obtidos do exercício prático sugerido na cadeira de Arquitetura de Computadores. O exercício consiste em criar um simulador de memória cache capaz de simular diversas arquiteturas.

## 2. Descrição do Problema

O exercício pede que seja implementado um simulador de cache capaz de ler um endereço e simular uma leitura real em uma cache. Ou seja, verifica-se se o endereço está contido na cache; caso não esteja, o bloco que contem esse endereço na memória principal deve ser trazido para a cache.

Foi fornecido um arquivo com 1012178 endereços em hexadecimal, cada um com 32 bits. Cada endereço deve ser lido pelo programa que deve ser capaz de contabilizar os erros e acertos para arquiteturas com:

- Tamanho da cache: 1024, 2048, 4096, 8192 e 16384 bytes;
- Tamanho do bloco: 16 bytes;
- Mapeamentos: direto, 2-way, 4-way e 8-way;
- Política de substituição: LRU e FIFO.

## 3. Solução

O simulador cache descrito nesse relatório foi feito em linguagem C e simula apenas uma arquitetura por vez. A entrada do programa são os parâmetros da cache como no exemplo a seguir:

```
./simulador 8192 8-way LRU
```

A cache foi implementada como uma s*truct* composta por dois ponteiros e quatro variáveis. Um ponteiro armazena os endereços das palavras contidas na cache e outro armazena o *timestamp* para cada linha da cache. As quatro variáveis indicam, respectivamente, o método de substituição(LRU ou FIFO) o tamanho da cache(em bytes), o número de linhas da cache e o tamanho do conjunto(no caso do mapeamento direto, o tamanho será 1), como ilustrado a seguir:

```
struct CACHE{
      unsigned int *in; //armazena os endereços
      unsigned int *t;  //timestamp
      short int substituition_mode;
      int size;
      int lines;
      int set;
};
```

Depois de definidos os parâmetros da cache, o arquivo é aberto e lido linha por linha. A cada leitura, uma função percorre o conjunto correspondente ao endereço na cache e verifica se já existe um endereço igual ao lido. As entradas da função são a cache, o endereço buscado. Caso já exista um endereço igual, a variável *hits* é acrescida de um, indicando que houve um acerto na cache. Caso não exista, a variável *errors* é acrescida de um e o programa segue para uma função responsável por simular um bloco sendo trazido da memória.

```
hit = find(cache, adress);

if(hit)
      hits++;
else{
      get_memory(cache, adress);
      errors++;
}
```

Para simular um bloco trazido da memória principal, na linha definida pelo algoritmo de substituição, é colocada a tag do endereço com os últimos 4 bits numerados de 1 a 16, um para cada palavra da linha. O trecho a seguir está na função que realiza esse procedimento:

```
tag = (adress >> 4);
tag = (tag << 4);

for(i = 0; i < LINE_SIZE; i++){ //LINE_SIZE = 16

      *(cache.in + line_adress*LINE_SIZE + i) = (tag + i);
}
```

Sempre que um endereço é buscado na cache, o *timestamp* é acrescido em um, mas apenas das linhas do conjunto que foi acessado. Como a comparação é feita apenas entre as linhas de cada conjunto, isso não trará problemas. No FIFO, sempre que uma linha recebe um bloco da memória principal, o seu *timestamp* é reduzido a um. No LRU, isso ocorre sempre que uma linha é acessada. A linha que possuir o maior *timestamp* é a linha que poderá ser substituída.

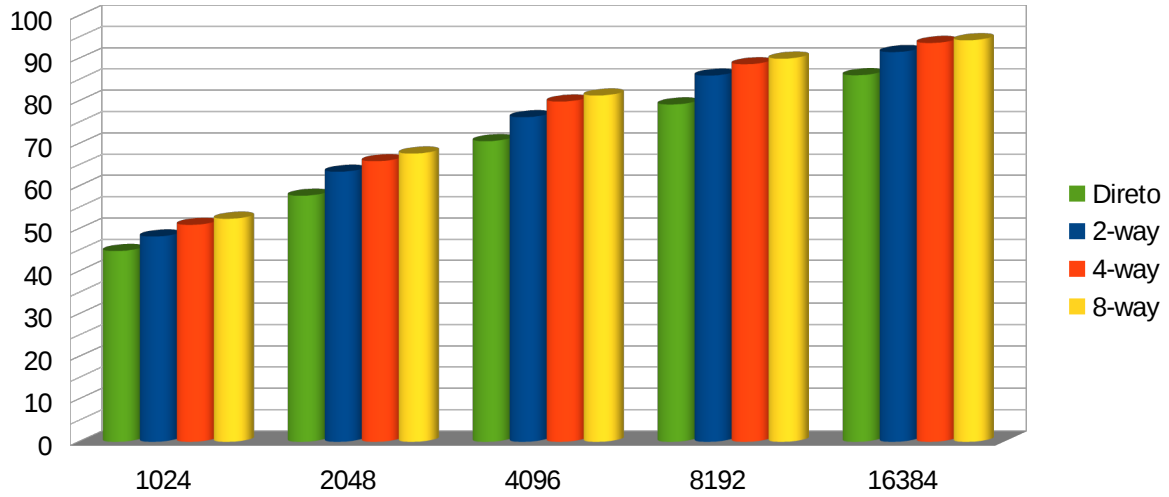
Feitos todos os procedimentos descritos, para cada arquitetura, os índices de acerto foram:

| | **1024** | **2048** | **4096** | **8192** | **16384** |
|---|---|---|---|---|---|
| **Direto** | 44.80% | 57.78% | 70.59% | 79.20% | 86.06% |
| **2-way~FIFO** | 46.48% | 61.53% | 74.44% | 84.57% | 90.48% |
| **4-way~FIFO** | 47.93% | 62.77% | 76.80% | 86.29% | 92.13% |
| **8-way~FIFO** | 48.65% | 63.87% | 77.52% | 87.18% | 92.55% |
| **2-way~LRU** | 48.20% | 63.41% | 76.21% | 85.97% | 91.49% |
| **4-way~LRU** | 50.94% | 65.89% | 79.88% | 88.64% | 93.63% |
| **8-way~LRU** | 52.36% | 67.67% | 81.30% | 89.96% | 94.28% |

## 4. Conclusão

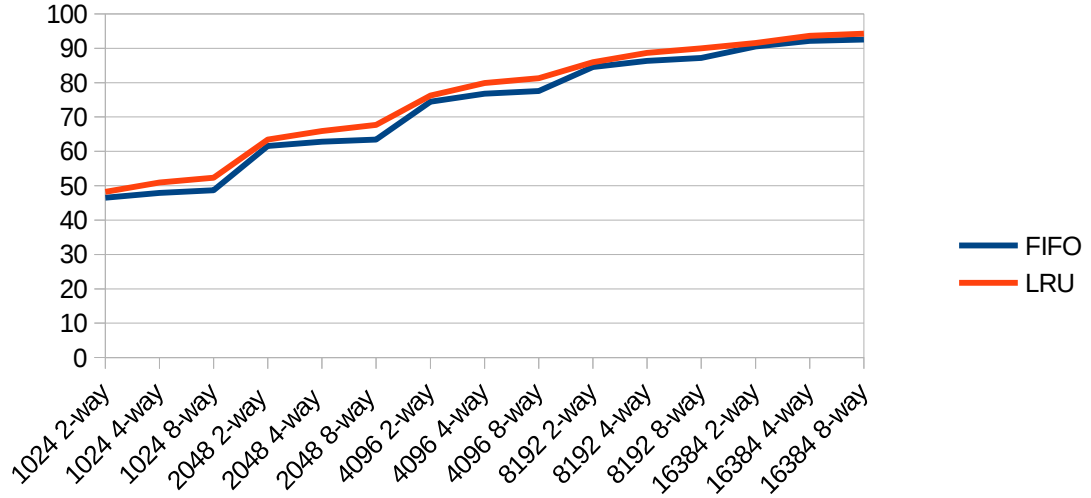
Dentre as muitas reflexões que podem ser feitas a partir dos resultados do simulador, podemos destacar dois:

**1.** Podemos observar que quanto maior o tamanho da cache e quanto maior o número de linhas no conjunto, maior o índice de acertos. Porém, esse aumento na taxa de acerto diminui conforme a cache fica a maior.




Como a memória cache é uma memória cara, os arquitetos de computadores devem buscar um ponto de equilíbrio, já que depois de certo ponto, aumentar o tamanho da cache gera ganhos mínimos de desempenho. O mesmo vale para o tamanho do conjunto, já que quanto mais linhas tiver o conjunto, mais complexo é o circuito para implementá-lo na cache.

**2.** Também se pode observar que o algoritmo de substituição LRU proporciona uma taxa de acerto um pouco melhor que o FIFO.




Isso se deve, provavelmente, ao fato de que o LRU explora melhor o princípio da localidade temporal. Para ilustrar melhor, suponha que, em uma cache que utiliza o FIFO, uma linha foi a primeira a entrar na cache. Mesmo que essa linha receba um acesso, ela poderá ser substituída no próximo acesso, o que pode levar a um erro na cache. No LRU, isso tem uma probabilidade menor de ocorrer, já que atualiza o *timestamp* a cada acesso.