Um Simulador que não é bem um simulador.

Trabalho realizado para a disciplina de Arquitetura e Organização de Computadores.

Do Curso de Engenharia da Computação.

Ministrado pelo Prof.Me. Rodrigo Porfírio da Silva Sacchi

Autor: Mateus Souza Silva

Dourados-MS
2022

Area de Importação dos Pacotes

In [ ]:
```
from math import log2  # Importando no pacote de matematica apenas o log na base 2.
import os  # Pacote que serve para utilizar os comando do SO,
#que utilizei posteriormente.
from recursos import *  # Arquivo que contém todas as funções de auxilio do projeto,
#que eu criei.
from random import randint  # Importando no pacote randômico apenas o
#método randômico de números inteiros
```

Função Principal

Por questão de boas práticas, criei a função principal do programa, por mais que não seja necessário para a organização é proveitoso fazer.

In [ ]:
```
def main():
  menu()
```

Função Menu Principal

Faz o controle de acesso do arquivo em que o usuario vai digitar o caminho dele, e como isso o programa já retira o nome do arquivo e o tipo de mapeamento utilizado, tratando os possiveis erros de entrada, para que nada de errado, e logo em seguida, chama a função que seleciona o tipo de mapeamento a partir do arquivo.

In [ ]:
```
def menu(arq_name='', tipo=''):

    print_menu(arq_name, tipo)
    op = input("Escolha uma das opções acima:")
```

```
    match op:
        case '1':
            path = input("Digite o diretório do arquivo:")
            if path != '':
                arq_name = os.path.basename(path)
                file, tipo = menu() if open_file(path) is None else open_file(path)
                cache = new_mc(file[2])
                if tipo == 'Associativo por Conjunto':
                    cache_set = new_mc_set(file[2], file[3])
                    print_menu(arq_name, tipo)
                    switch_mapping(arq_name, tipo, file, cache, cache_set)
                else:
                    switch_mapping(arq_name, tipo, file, cache)

            else:
                print('Diretório Inválido')
                menu()
                input()
                os.system('cls'if os.name == 'nt' else 'close')
                # Função que pega o nome do SO e faz uma operação condicional
                #para saber qual comando de limpeza vai executar,
                #caso for 'nt' que seria o windows, vai ser 'cls' o comando, caso não 'c
        case 'S' | 's' | 'n' | 'N':
            print('Saindo...')
            exit()
```

Função de Escolha de Mapeamento

É a função que a partir do arquivo faz o controle de qual mapeamento vai ser escolhido, e passando todos os parametros que cada um precisa. Utilizada para uma maior organização e distribuição de casos especificos como o Associativo por Conjunto que é outra idéia de cache que foi feito.

In [ ]:
```
def switch_mapping(arq_name='', tipo='', file=None, cache=None, cache_set=None):
    match tipo:
        case 'Direto':
            os.system('cls' if os.name == 'nt' else 'close')
            print('Mapeamento Direto')
            direct_mapping(file, cache, name=arq_name, tp=tipo)
            menu(arq_name, tipo)
        case 'Associativo':
            os.system('cls' if os.name == 'nt' else 'close')
            print('Mapeamento Associativo')
            associative_mapping(file, cache, name=arq_name, tp=tipo)
        case 'Associativo por Conjunto':
            os.system('cls' if os.name == 'nt' else 'close')
            print('Mapeamento Associativo por Conjunto')
            set_associative(file, cache_set, name=arq_name, tp=tipo)
```

Mapeamento Direto

A ideia é que o usuario digite um endereço da MP, e chame uma função que divide bit a bit esse endereço Exemplo: Levando em consideração um MP de **64** palavras 2^**6** palavras, temos que o tamanho do endereço é de **6** bits Tomando o **15** de exemplo, o seu binario é **1111**, transformando com quantidade de bits do endereço temos **001111**. Esse é o primeiro passo, segundo, cada bit é convertido em um item de lista, o número **001111 -> [0,0,1,1,1,1]**. Depois desse processo fica muito simples de separar os bits para cada coisa, tomando como **tag** = 2bits, **r** = 3bits e **w** = 1bit, usando o recurso de recortes do python

podemos utilizar para descobrir a **tag**: `endereço[:tag] - > [0,0]` . E assim com os outros, **r** = `endereço[tag:-w] - > [1,1,1]` e **w** = `endereço[:-w] - > [1]` .

Para que caia na posição correta da cache, eu utilizei de alguns recursos do python, peguei o r da lista transformei em decimal, e com esse decimal usei de indice na lista, no mesmo compara as tag, se forem iguais acerto, se não falha.

Vale ressaltar que em todos os mapeamentos as funções são recursivas, apenas por ser mais elegante e mais limpo que o iterativo.

In [ ]:
```
def direct_mapping(dados, cache, fim='', acerto=0.0, falha=0.0, total=0.0, name='', tp='
    if fim == 'sair':
        print('Total de acertos e falhas')
        print_miss_hit(acerto, falha, total)
        input()
        os.system('cls' if os.name == 'nt' else 'close')
        menu(name, tp)
    else:
        ender = treatment_input(dados)
        mc = cache
        address = int(log2(dados[0]))
        s = int(log2(dados[0]/dados[1]))
        r = int(log2(dados[2]))
        w = int(log2(dados[1]))
        tag = s - r
        binario, result = slice_bin(address, ender)

        print(f'Endereço :{binario}')
        print('S:', result[:s])
        print('Tag:', result[:tag], 'R:', result[tag:-w] if w > 0 else result[r:],
              'W:', result[-w:] if w > 0 else 0)

        linha = result[tag:-w] if w > 0 else result[r:]
        linha = [str(i) for i in linha]
        r_linha = "".join(map(str, linha))
        pos_linha = int(bin_to_decimal(r_linha))

        if mc[pos_linha][:] == 'Vazia' or mc[pos_linha][0] != result[:tag]:
            mc[pos_linha][:] = result[:tag], 'dados'
            falha += 1
            total += 1
        elif mc[pos_linha][0] == result[:tag]:
            acerto += 1
            total += 1

        print_cache(cache, pos_linha, r)

        print_miss_hit(acerto, falha, total)

        fim = input('Digite (sair) pra terminar o mapeamento ou enter para continuar map
        os.system('cls' if os.name == 'nt' else 'close')
        direct_mapping(dados, cache, fim, acerto, falha, total, name, tp)
```

Mapeamento Totalmente Associativo

A principio a ideia é a mesma do direto do fatiamento do endereço e armazenando em uma lista, a diferença que no associativo não existe mais os bits para r, sendo assim s é igual a tag, com isso não temos mais uma linha predeterminada da cache para colocarmos o dados, e ele vai ser escrito primeira linha vazia

que ele encontrar. Depois que a cache se enche, utilizei o pacote random do python, sendo mais especifico o método `randint()`, que gera um número inteiro randomico a partir de range, e com esse valor substitui na cache essa posição, caso a tag não esteja já na cache.

In [ ]:

```
def associative_mapping(dados, cache, fim='', init=0, quant=0,
                        acerto=0.0, falha=0.0, total=0.0, name='', tp=''):

    def cache_cheia(ini):
        if ini == int(dados[2]):
            return True
        return False

    def tag_compare(tg, cache_busca):
        for i in range(len(cache_busca)):
            if cache_busca[i][0] == tg:
                return i
        return -1

    if fim == 'sair':
        print('Total de acertos e falhas')
        print_miss_hit(acerto, falha, total)
        input()
        os.system('cls' if os.name == 'nt' else 'close')
        menu(name, tp)
    else:
        mc = cache
        address = int(log2(dados[0]))
        tam_cache = dados[2]
        tag = int(log2(dados[0] / dados[1]))
        w = int(log2(dados[1]))
        ender = treatment_input(dados)
        binario, result = slice_bin(address, ender)
        busca = tag_compare(result[:tag], cache)

        print(f'Endereço :{binario}')
        print('S/Tag:', result[:tag], 'W:', result[-w:] if w > 0 else 0)

        if busca != -1:
            acerto += 1
            total += 1
            print('Acerto')
            print_cache(mc, busca, tam_cache, tp)
        elif not cache_cheia(quant):
            mc[init][:] = result[:tag], 'dados'
            quant += 1
            falha += 1
            total += 1
            print_cache(mc, init, tam_cache, tp)
            if init == tam_cache-1:
                init = 0
            else:
                init += 1
        else:
            init = randint(0, tam_cache-1)
            mc[init][:] = result[:tag], 'dados'
            falha += 1
            total += 1
            print_cache(mc, init, tam_cache, tp)

        print_miss_hit(acerto, falha, total)

        fim = input('Digite (sair) pra terminar o mapeamento ou enter para continuar map
        os.system('cls' if os.name == 'nt' else 'close')
```

```
        associative_mapping(dados, cache, fim, init, quant, acerto,
                            falha, total, name, tp)
```

Mapeamento Associativo por Conjunto

Basicamente a mistura do mapeamento direto com o mapeamento associativo, olhando o código a idéia é muito parecida com a do direto com uma diferença apenas, que cada conjunto têm uma quantidade de linhas, que a conta para isso é simples Linhas_cache ÷ qtd_conjuntos = *qtd_linhas_por_conjunto*, sendo que qtd_conjunto ≤ Linhas_cache. Apesar de não ter necessidade, eu implementei apenas um algoritmo de substituição, por ser o mais simples e util, que foi o FIFO(*First in, First out*), que o primeiro dado que entrou na cache, é o primeiro a sair, e é sempre verificado se a tag já não existe na cache, antes de inserir qualquer novo valor. Nesse mapeamento em especifico, tive que uma cache especial para ele, a solução mais prática que eu encontrei foi usar uma lista tridimensional, como o binário de endereço do meu mapeamento é em lista acabou que fez necessário a lista tridimensional. Exemplo: `cache = [conjunto1[linhas_conj1[bits]], conjunto2[linhas_conj2[bits]]]` A lista mais externa seria a cache, a lista intermediária seria os conjuntos, e a lista mais interna os bits de endereço de cada linha. Fica assim a cache no python:

lista_mais_externa [lista_intermediaria [lista_mais_interna [0,0,0,0,0],[0,0,0,0,1]],[[0,0,0,1,0],[0,0,0,1,1]]].

In [ ]:

```
def set_associative(dados, cache, fim='', fifo=0,
                    acerto=0.0, falha=0.0, total=0.0, name='', tp=''):
    if fim == 'sair':
        print('Total de acertos e falhas')
        print_miss_hit(acerto, falha, total)
        input()
        os.system('cls' if os.name == 'nt' else 'close')
        menu(name, tp)
    else:
        ender = treatment_input(dados)
        mc = cache
        address = int(log2(dados[0]))
        s = int(log2(dados[0] / dados[1]))
        tam_cache = dados[2]
        d = int(log2(dados[3]))
        w = int(log2(dados[1]))
        tag = s - d
        binario, result = slice_bin(address, ender)

        print(f'Endereço :{binario}')
        print('S:', result[:s])
        print('Tag:', result[:tag], 'D:', result[tag:-w] if w > 0 else result[d:],
              'W:', result[-w:] if w > 0 else 0)
        print(tam_cache)
        linha = result[tag:-w] if w > 0 else result[d:]
        linha = [str(i) for i in linha]
        r_linha = "".join(map(str, linha))
        pos_linha = int(bin_to_decimal(r_linha))
        for i in range(tam_cache):
            if mc[pos_linha][i] == 'Vazia':
                mc[pos_linha][i] = result[:tag]
                falha += 1
                total += 1
                break
            elif mc[pos_linha][i] == result[:tag]:
                acerto += 1
                total += 1
```

```
            elif 'Vazia' not in mc[pos_linha]:
                mc[pos_linha][fifo] = result[:tag]
                if fifo == (tam_cache-1):
                    fifo = 0
                else:
                    fifo += 1
                break

        print_cache(cache, pos_linha, d)

        print_miss_hit(acerto, falha, total)

        fim = input('Digite (sair) pra terminar o mapeamento ou enter para continuar map
        os.system('cls' if os.name == 'nt' else 'close')
        set_associative(dados, cache, fim, fifo,
                        acerto, falha, total, name, tp)
```

Arquivo de recurso

Como eu utilizei esse ambiente apenas pra deixar os comentários melhor de enteder e até mais elegantes, não vou separar em dois arquivos, e sim separar em Tópico o arquivo de recurso.

Tópico que possui todas as função que têm mais a ver com recursos necessários para tratamento de dados, converção, comparações e prints.

Para que o projeto em si ficasse mais organizado, foi separado dessa forma as funções.
Com isso, neste Tópico contém apenas as funções de auxilio.

Cores para deixar o print mais intuitivo.

In [ ]:
```
certo = "\033[92m"
reset = '\033[0m'
aviso = '\033[91m'
```

Função para printar o menu já todo formatado corretamente, mostrando o arquivo carregado e tipo de mapeamento

---

In [ ]:
```
def print_menu(arq_name='', tipo=''):
    if arq_name == '' and tipo == '':
        arq_name = 'Nenhum arquivo carregado'
        tipo = 'Nenhum mapeamento ativo'
    print("-"*69)
    print("|" + " "*23, "* Menu Principal *" + " "*24, "|")
    print("-" * 69)
    print(f"|-> Carregar Arquivo  1 | Arquivo Carregado:{arq_name:<24}|")
    print(f"|-> Tipo de Mapeamento:{tipo}")
    print("|-> Sair [S/n]"+" "*54+"|")
    print("-" * 69)
```

Função que abre o arquivo em modo de leitura, retira os

dados e armazena em uma lista.

Usando um for para percorrer o arquivo retirando o \n dos dados e convertendo em inteiro**

Em um caso especial do Associativo por Conjunto em que busca : para separar a quantidade de linha por conjunto e a quantidade de conjunto.

In [ ]:
```
def open_file(path):
    arq = open(path, 'r')
    if arq is None:
        print('Não foi possível abrir o arquivo')
        return None
    tipo = ''
    dados = []
    for line in arq:
        if ':' in line:
            aux = line.split(':')
            if aux[1] > aux[0]:
                raise Exception('Número de conjuntos maior que o número de linhas da cac
                                'Mude no arquivo por favor')
            dados.append(int(aux[0]))
            dados.append(int(aux[1].strip('\n')))
            continue
        if '\n' in line:
            dados.append(int(line.strip('\n')))
        else:
            tipo = line

    return dados, tipo
```

Cria uma lista de lista para representar a cache.

Com um conceito chamado list comprehensions que basicamente faz um for dentro da lista colocando outra lista com um valor 'Vazio' para representar uma linha da cache não preenchida, que foi necessário para fazer do jeito que foi projetado a cache, que era de considerar o endereço em binário como uma lista separando bit a bit.

In [ ]:
```
def new_mc(line):
    new_cache = [['Vazio'] for _ in range(line)]
    return new_cache
```

Cria uma lista tridimensional para representar a cache no mapeamento Associativo por Conjunto.

Na forma que eu projetei, todos os mapeamentos baseado em lista o endereço em binario, acabou ficando uma lista Tridimensional, para representar a Cache, o Conjunto e a linha.

In [ ]:
```
def new_mc_set(line, mc_set):
    if mc_set is not None:
        new_cache_set = [['Vazia' for _ in range(line/mc_set)] for _ in range(mc_set)]
        return new_cache_set
```

Função para converter um número binario para decimal.

In [ ]:
```
def bin_to_decimal(n):
    return int(n, 2)
```

Função que recebe um endereço fornecido por parametro na função e completa com a quantidadede bits do endereço da MP.

In [ ]:
```
def quant_bin(qtd, num):
    binario = str(bin(num).removeprefix('0b'))
    binario = ((qtd - len(binario)) * '0') + binario

    return binario
```

Função de recorte de endereço

É passando por parametro um endereço e o tamanho do enderço da MP, e como é uma função recorrente, primeiro converte em binario e adiciona 0 que faltam para atingir os bits necessários de endereço, e depois faz um processo de separar bit a bit em uma lista, para que fique mais facil de organizar cada sessão.

In [ ]:
```
def slice_bin(address, ender):
    a = 1
    result = []
    binario = quant_bin(address, ender)
    for i in range(0, len(binario), a):
        # Converte para inteiro, depois do processo de fatiamento
        result.append(int(binario[i: i + a]))

    return binario, result
```

Função de Tratamento

O intuito dessa função é apenas de tratar a entrada do usuário para que ele não digite um número de endereço maior que a MP permite, usando de uma técnica chamada Expressão Condicional, que é muito similar a ideia do Operador Ternario, que faz um if e else em uma linha.

In [ ]:
```
def treatment_input(dados):
    ender = -1
    while ender < 0 or ender > (dados[0] - 1):
        ender = int(input(f'Digite o endereço entre 0 e {dados[0] - 1}:'))
        if ender < 0 or ender > (dados[0] - 1)
        else print('Você digitou um número fora da faixa, por favor')
    return ender
```

Um função print que têm como objetivo mostrar a linha da cache endereçada, duas antes e duas após.

Como o print é comum aos 3 mapeamentos, como via de praticidade foi feita uma função para englobar a todos.
No caso do Associativo, que têm uma idéia diferente por ser aleatório, o print dele é diferente dos outros, para que possa ser impresso corretamente ele, o `print_cache` recebe o tipo como parâmetro e verifica se é o Associativo para fazer essa distinção.

In [ ]:
```
def print_cache(cache, pos, tam, tp=''):
    tipo = 'associativo'
    if tp.casefold() == tipo.casefold():
        print('Tipo:', tp)
        print('Posição | Tag e Conteudo')
        print(pos)
        print(bin(pos - 2).removeprefix('0b'), ':',
              cache[pos - 2] if pos - 2 >= 0 else aviso + 'Fora da linha' + reset)
        print(bin(pos - 1).removeprefix('0b'), ':',
              cache[pos - 1] if pos - 1 >= 0 else aviso + 'Fora da linha' + reset)
        print(certo, (bin(pos).removeprefix('0b')), ':', cache[pos], reset)
        print(bin(pos + 1).removeprefix('0b'), ':',
              cache[pos + 1] if pos + 1 <= tam - 1 else aviso + 'Fora da linha' + reset)
        print(bin(pos + 2).removeprefix('0b'), ':',
              cache[pos + 2] if pos + 2 <= tam - 1 else aviso + 'Fora da linha' + reset)
    else:
        print('Posição | Tag e Conteudo')
        print(quant_bin(tam, pos - 2) + 4 * ' ' + ':',
              cache[pos - 2] if pos > 1 else aviso + 'Fora da linha' + reset)
        print(quant_bin(tam, pos - 1) + 4 * ' ' + ':',
              cache[pos - 1] if pos > 0 else aviso + 'Fora da linha' + reset)
        print(certo + (quant_bin(tam, pos)) + 4 * ' ' + ':', cache[pos], reset)
        print(quant_bin(tam, pos + 1) + 4 * ' ' + ':',
              cache[pos + 1] if pos + 1 <= (2 ** tam) - 1 else aviso + 'Fora da linha' +
        print(quant_bin(tam, pos + 2) + 4 * ' ' + ':',
              cache[pos + 2] if pos + 2 <= (2 ** tam) - 1 else aviso + 'Fora da linha' +
```

Print Acerto e Falha

Todos os mapeamentos possuem o acerto e falha, então para evitar repetição de código, foi feita essa função para que evita-se isso e também para que seja padronizada todos as impressões em todos os mapeamentos.

In [ ]:
```
def print_miss_hit(acerto, falha, total):
    if total > 0:
        print(f'Falha:{(falha / total) * 100:0.2f}%\tAcerto:{(acerto / total) * 100:0.2f
```

Chamando a Função Principal

In [ ]:
```
main()
```

Finalizado