TP 7

Pour aller plus loin

Avant de commencer, sachez que chacune des fonctions ci-dessous comportent des commentaires pour expliquer leurs fonctionnement.

Gestion des synonymes dans la cache *direct-mapped*

Voici la fonction in_cache_direct_mapped_fixed que j'ai codé:

```python
def in_cache_direct_mapped_fixed(mem_class, adresse):
    binary = str(bin(adresse)).split("0b")[1] # On récupére le code binaire
    tag = binary[:-2] if binary[:-2] != '' else 0 # On récupère le tag

    if adresse > len(mem_class):
        adresse = len(mem_class) - 1

    value = get_value(mem_class, adresse)

    # Si la valeut est OK et que le tag est valide on fini
    if value["ok"] and int(value["tag"]) == int(value["tag"]):
        return "HIT", value["data"]

    return "MISS", None
```

et voici les résultats:

```
kakesinfo@laptop:~/Workspace/I22/TP 7$ python3 tp.py
('HIT', 0) 0
('HIT', 119) 7
('MISS', None) 1
('HIT', 119) 3
kakesinfo@laptop:~/Workspace/I22/TP 7$
```

Je n'ai pas réussi à comprendre pourquoi mon 4ème résultat ne correspond pas à l'exemple attitré.

Politique de remplacement aléatoire

Voici la fonction replace_random que j'ai codé:

```python
def replace_random(mem_asso, mem):
    # On liste chaque entrée de la mémoire et son indice,
    # cela nous permet de vérifier si l'entrée est OK
    # et dans le cas contraire on retourne l'indice
    for index, entry in enumerate(mem):
        if not entry["ok"]: return index

    # dans le cas ou toutes les entrées sont OK
    # on retourne un indice aléatoire
    return randint(0, len(mem) - 1)
```

et voici les résultats qui correspondent aux exemples donnés dans l'énoncé:

```
kakesinfo@laptop:~/Workspace/I22/TP 7$ python3 t
1
3
```

- Le 1 correspond à l'indice de la première entrée non valide du premier exemple
- Le 3 est une valeur aléatoire dans le cas ou toutes les entrées sont valides

Politique de remplacement FIFO

1. add_fifo

Voici la fonction add_fifo que j'ai codé:

```python
def add_fifo(fifo, valeur):
    # On cherche l'indice du premier None
    noneIndex = 0
    while noneIndex < len(fifo) and fifo[noneIndex] != None:
        noneIndex += 1
    # On vérifie si la file est pleine
    if noneIndex >= len(fifo): return False
    # Si elle ne l'est pas on ajoute la valeur
    else:
        fifo[noneIndex] = valeur
        return fifo
```

avec les résultats valides:

```
kakesinfo@laptop:~/Workspace/I22/TP 7$ python3 tp.py
[None, None, None, None]
[10, None, None, None]
[10, 99, None, None]
[10, 99, 2, None]
```

2. get_fifo

Voici la fonction get_fifo que j'ai codé:

```python
def get_fifo(fifo):
    # On récupère et enlève la valeur à l'indice 0
    valeur = fifo.pop(0)
    # On ajoute un None à la fin de la file
    fifo.append(None)
    return fifo, valeur
```

et voici les résultats qui correspondent aux exemples donnés.

```
kakesinfo@laptop:~/Workspace/I22/TP 7$ python3 tp.py
[10, 99, 2, None]
[99, 2, None, None] 10
[2, None, None, None] 99
[None, None, None, None] 2
```

3. replace_fifo

```python
def replace_fifo(mem, fifo):
    # On trouve l'indice de la première entrée
    # de la mémoire qui est invalide
    invalideIndex = None
    for index, entry in enumerate(mem):
        if not entry["ok"]:
            invalideIndex = index
            break

    #SI une entrée est invalide
    # on retourne son indice et la nouvelle file
    if invalideIndex != None:
        return invalideIndex, add_fifo(fifo, invalideIndex)
    # SInon on récupère la plus ancienne valeur
    # et on la remet au dessus del a file
    else:
        valeur = get_fifo(fifo)[1]
        fifo = add_fifo(fifo, valeur)
        return valeur, fifo
```

```
kakesinfo@laptop:~/Workspace/I
(1, [2, 3, 0, 1])
(2, [3, 0, 1, 2])
(3, [0, 1, 2, 3])
```

Politique de remplacement LRU

Lorsque j'ai codé cette fonction, je me suis un peu inspiré de celle de add_fifo pour la partie grace à laquelle on trouve l'indice du dernier nombre en cache.

```python
def update_lru(pile, valeur):
    # On cherche l'indice du premier None
    noneIndex = 0
    while noneIndex < len(pile) and pile[noneIndex] != None:
        noneIndex += 1
    noneIndex -= 1 #On trouve l'indice du dernier nombre en ca

    # On cherche l'indice de la valeur a mettre àjour
    index = 0
    while pile[index] != valeur:
        index += 1

    pile.pop(index) # On enlève la valeur
    pile.pop(noneIndex) # On enlève le None
    pile.append(valeur) # On ajoute la valeur à la fin
    pile.append(None) # 0, rajoute le None enlevé

    return pile
```

```
kakesinfo@laptop:~/Workspace/I22/TP 7$ python3 tp.py
[2, 0, 3, None]
[0, 3, 2, None]
[0, 3, 2, None]
kakesinfo@laptop:~/Workspace/I22/TP 7$
```