CS/ECE 752
Fall 2024
Homework 4 SOLUTION
Due 11 AM Central Time on Saturday, October 19th, 2024

NAME: ______________________________

You should do this assignment on your own, although you are encouraged to talk with classmates electronically or on Piazza about any issues you may have encountered. The standard late assignment policy applies: you may submit up to 1 day late with a 10% penalty.

What to Hand In

To submit your assignment, please type up your answers to the following questions and submit **one PDF named HW4-<netID>.pdf** on Canvas. If you prefer writing your answers by hand, that is fine, but please scan your solutions and submit them on Canvas. **Please make sure to put your name in the above prompt of the first page of the PDF.** *If at all possible, please retain the format of the provided PDF/Word document. I am going to grade this homework using Gradescope, which relies on answers being in set places.*

Total Points: 57

In this assignment we will be examining how different cache replacement policies behave on relatively small, simple traces. Some of these replacement policies may be ones you've seen in prior classes (e.g., LRU) while others are more complex (e.g., SRRIP) and relatively new.

You may assume the following:

- We have a 512B cache with 64B blocks and 4-way set associativity.
- You can assume the cache is PIPT and the translation has already been done for you (Note: we will not discuss virtual vs. physical addressing before this assignment is due – essentially PIPT means you can assume the provided addresses (below) are the addresses the cache uses for accesses).
- All addresses are 64-bits.
- All of the requests are loads.
- There are no other cores in the system, so these are the only memory accesses (i.e., no interleaved memory requests from another core).
- We are using standard hexadecimal/binary notation. So 0xABCD = $(1010\ 1011\ 1100\ 1101)_2$
- The data values can be ignored.

- All replacement policies will evict an invalid entry first (before any valid entry), and will do so in monotonically increasing order (i.e., way0, then way1, and so on) **except for PLRU**.
- Initial replacement values:
    1. For all replacement policies other than BRRIP/SRRIP, you can assume that way 0 in a given set is the initial replacement value (e.g., LRU would initially want to replace way 0 in a given index).
    2. For BRRIP/SRRIP, assume $M = 2$ (i.e., max value = 2^2-1 = 3) and the initial RRIP values as shown/discussed below.

Problem 1 [3 points]

Which bits are used for the offset, set index, and tag? To receive credit, you must show your work.

Solution:

The prompt says we have 64-bit addresses, 4-way set associativity, a 512B cache, and 64B blocks. Thus, we have all the information we need to determine which bits are offset, set index, and tag.

Offset: $\log_2(64) = 6$

Thus we need 6 bits for the offset into a given cache line, and since offset bits are the least significant bits, we use bits 5:0 for the offset

Set Index: $\frac{\frac{512\ B}{64\ B/line}}{4\ ways} = 2\ \frac{cache\ lines}{way}$

$\log_2(2) = 1$

Thus we need 1 bit for the set Way (since there are 2 cache lines per way). Since we used bits 5:0 for the offset and Way bits come next, that means we use bit 6 for set indexing.

Tag: Tag is always the remaining bits after Way and offset. Since we have 64 bits:

$$64 - 6 - 1 = 57\ bits$$

Thus 57 bits (and bits 63:7) are used for the tag.

Problem 2 [54 points]

For all cache replacement policies, **assume the following access pattern** (you can assume all bits that are not shown are 0):

```
LD 0x0     // Access #0
LD 0x81    // Access #1
LD 0x100   // Access #2
LD 0x188   // Access #3
LD 0x4     // Access #4
LD 0x200   // Access #5
LD 0x18C   // Access #6
LD 0x108   // Access #7
LD 0x380   // Access #8
LD 0x8     // Access #9
```

Given the above information and access pattern, fill in the rest of the tables for the LFU, LRU, PseudoLRU (Tree-PLRU), SecondChance, SRRIP, and BRRIP replacement policies. The format for the cache contents are as follows: Way 0 [way0, way1, way2, way3], Way 1 [way0, way1, way2, way3]. If there is no victim, please use "N/A" or "X" in that column, and if the victim is an invalid entry make clear which way is the victim – **do not leave these blank**. **Also note that "---" means invalid for the cache contents and not needed for the address, hit/miss, and victim columns.**

Note: Every access will go to set index 0 (because bit 6 is always 0 for all addresses), so set index 1 is unused in all replacement policies.

LFU:

Note: if needed see slide 139 in Unit 10 for LFU details. Here <A,B,C,D> represents the usage counter for each way of a given set index.

Cache Contents	LFU Vals	Access Address	LFU Hit/Miss	Victim (If Any)
[--, --, --, --], [--, --, --, --]	[<0,0,0,0>; <0,0,0,0>]	0x0	Miss	Way 0 (--)
[**0x0**, --, --, --], [--, --, --, --]	[<**1**,0,0,0>; <0,0,0,0>]	0x81	Miss	Way 1 (--)
[0x0, **0x80**, --, --], [--, --, --, --]	[<1,**1**,0,0>; <0,0,0,0>]	0x100	Miss	Way 2 (--)
[0x0, 0x80, **0x100**, --], [--, --, --, --]	[<1,1,1,0>; <0,0,0,0>]	0x188	Miss	Way 3 (--)
[0x0, 0x80, 0x100, **0x180**], [--, --, --, --]	[<1,1,1,**1**>; <0,0,0,0>]	0x4	Hit	N/A
[**0x0**, 0x80, 0x100, 0x180], [--, --, --, --]	[<**2**,1,1,1>; <0,0,0,0>]	0x200	Miss	Way 1 (0x80)
[0x0, **0x200**, 0x100, 0x180], [--, --, --, --]	[<2,**1**,1,1>; <0,0,0,0>]	0x18C	Hit	N/A
[0x0, 0x200, 0x100, **0x180**], [--, --, --, --]	[<2,1,1,**2**>; <0,0,0,0>]	0x108	Hit	N/A
[0x0, 0x200, **0x100**, 0x180], [--, --, --, --]	[<2,1,**2**,2>; <0,0,0,0>]	0x380	Miss	Way 1 (0x200)
[0x0, **0x380**, 0x100, 0x180], [--, --, --, --]	[<2,**1**,2,2>; <0,0,0,0>]	0x8	Hit	N/A
[**0x0**, 0x380, 0x100, 0x180], [--, --, --, --]	[<**3**,1,2,2>; <0,0,0,0>]	---	---	---

LRU:

Cache Contents	LRU Way	Access Address	LRU Hit/Miss	Victim (If Any)
[--, --, --, --], [--, --, --, --]	[0, 0]	0x0	Miss	Way 0 (--)
[**0x0**, --, --, --], [--, --, --, --]	[**1**, 0]	0x81	Miss	Way 1 (--)
[0x0, **0x80**, --, --], [--, --, --, --]	[**2**, 0]	0x100	Miss	Way 2 (--)
[0x0, 0x80, **0x100**, --], [--, --, --, --]	[**3**, 0]	0x188	Miss	Way 3 (--)
[0x0, 0x80, 0x100, **0x180**], [--, --, --, --]	[**0**, 0]	0x4	Hit	N/A
[**0x0**, 0x80, 0x100, 0x180], [--, --, --, --]	[**1**, 0]	0x200	Miss	Way 1 (0x80)
[0x0, **0x200**, 0x100, 0x180], [--, --, --, --]	[**2**, 0]	0x18C	Hit	N/A
[0x0, 0x200, 0x100, **0x180**], [--, --, --, --]	[2, 0]	0x108	Hit	N/A
[0x0, 0x200, **0x100**, 0x180], [--, --, --, --]	[**0**, 0]	0x380	Miss	Way 0 (0x0)
[**0x380**, 0x200, 0x100, 0x180], [--, --, --, --]	[**1**, 0]	0x8	Miss	Way 1 (0x200)
[0x380, **0x0**, 0x100, 0x180], [--, --, --, --]	[**3**, 0]	---	---	---

Pseudo (Tree) LRU (PLRU):

Note: because we have 4-way set associativity, for Tree PLRU we can model the replacement information with {*R; A; B*} where *R* represents the value of the root of the tree, *A* represents the value of the left subtree, and *B* represents the value of the right subtree. *R*, *A*, and *B* are either 0 or 1 at all times. If needed see slides 12 and 137-138 in Unit 10 for an example of this.

Cache Contents	PLRU Vals	Access Address	PLRU Hit/Miss	Victim (If Any)
[--, --, --, --], [--, --, --, --]	[{0; 0; 0}, {0; 0; 0}]	0x0	Miss	Way 0 (--)
[**0x0**, --, --, --], [--, --, --, --]	[{**1; 1;** 0}, {0; 0; 0}]	0x81	Miss	Way 2 (--)
[0x0, --, **0x80**, --], [--, --, --, --]	[{**0**; 1; **1**}, {0; 0; 0}]	0x100	Miss	Way 1 (--)
[0x0, **0x100**, 0x80, --], [--, --, --, --]	[{**1; 0;** 1}, {0; 0; 0}]	0x188	Miss	Way 3 (--)
[0x0, 0x100, 0x80, **0x180**], [--, --, --, --]	[{**0**; 0; **0**}, {0; 0; 0}]	0x4	Hit	N/A
[**0x0**, 0x100, 0x80, 0x180], [--, --, --, --]	[{**1; 1;** 0}, {0; 0; 0}]	0x200	Miss	Way 2 (0x80)
[0x0, 0x100, **0x200**, 0x180], [--, --, --, --]	[{**0**; 1; **1**}, {0; 0; 0}]	0x18C	Hit	N/A
[0x0, 0x100, 0x200, **0x180**], [--, --, --, --]	[{**0**; 1; **0**}, {0; 0; 0}]	0x108	Hit	N/A

[0x0, **0x100**, 0x200, 0x180], [--, --, --, --]	[{**1; 0;** 0}, {0; 0; 0}]	0x380	Miss	Way 2 (0x200)
[0x0, 0x100, **0x380**, 0x180], [--, --, --, --]	[{**0;** 0; **1**}, {0; 0; 0}]	0x8	Hit	N/A
[**0x0**, 0x100, 0x380, 0x180], [--, --, --, --]	[{**1; 1;** 1}, {0; 0; 0}]	---	---	---

0x4 Hit: Update both levels of tree to point away from new MRU (0x0 – way 0) – but leave the other leaf node of subtree alone (i.e., only update my leaf and parent node(s) above me). This flips my leaf subtree from 0 → 1 and my parent from 0 → 1 – hence {1; 1; 0} is the result.

0x18C Hit: Root node (my parent) is already pointing way from my leaf, so leave it that way. Update my leaf to point away from new MRU (way 3) – hence {0; 1; 0} is the result.

0x108 Hit: Update root node (0 → 1) and my leaf (1 → 0) to point away from new MRU (way 1). Thus {1; 0; 0} is the result.

0x8 Hit: Update root node (0 → 1) and my leaf (0 → 1) to point away from new MRU (way 0). Thus {1; 1; 1} is the result.

Second Chance (SC):

For second chance replacement, we use a FIFO-like replacement policy **except** we track if the line has been accessed (aka touched) at least once since it was inserted into the cache. If so, when deciding which line to evict we give a line with this "touched" flag set a second chance before evicting it. Thus, we need an additional bit of state per cache line tracking whether the line has been touched or not since it was inserted. To represent this, we denote this as <FIFO Index>[way0, way1, way2, way3] for each Way in the cache. For example, <0>[1, 1, 1, 0]; <1>[0, 1, 1, 1] would indicate that for set Way 0: way0 is currently in FIFO position and way0/1/2 have all been touched since they were inserted.

Cache Contents	SC Vals.	Access Address	SC Hit/Miss	Victim (If Any)
[--, --, --, --], [--, --, --, --]	<0>[0,0,0,0], <0>[0,0,0,0]	0x0	Miss	Way 0 (--)
[**0x0**, --, --, --], [--, --, --, --]	<**0**>[**0**,0,0,0], <0>[0,0,0,0]	0x81	Miss	Way 1 (--)
[0x0, **0x80**, --, --], [--, --, --, --]	<0>[0,**0**,0,0], <0>[0,0,0,0]	0x100	Miss	Way 2 (--)
[0x0, 0x80, **0x100**, --], [--, --, --, --]	<0>[0,0,**0**,0], <0>[0,0,0,0]	0x188	Miss	Way 3 (--)
[0x0, 0x80, 0x100, **0x180**], [--, --, --, --]	<0>[0,0,0,**0**], <0>[0,0,0,0]	0x4	Hit	N/A
[**0x0**, 0x80, 0x100, 0x180], [--, --, --, --]	<0>[**1**,0,0,0], <0>[0,0,0,0]	0x200	Miss	Way 1 (0x80)
[0x0, **0x200**, 0x100, 0x180], [--, --, --, --]	<0>[**0**,**0**,0,0], <0>[0,0,0,0]	0x18C	Hit	N/A
[0x0, 0x200, 0x100, **0x180**], [--, --, --, --]	<0>[0,0,0,**1**], <0>[0,0,0,0]	0x108	Hit	N/A
[0x0, 0x200, **0x100**, 0x180], [--, --, --, --]	<0>[0,0,**1**,1], <0>[0,0,0,0]	0x380	Miss	Way 0 (0x0)

[**0x380**, 0x200, 0x100, 0x180], [--, --, --, --]	<2>[0,0,1,1], <0>[0,0,0,0]	0x8	Miss	Way 1 (0x200)
[0x380, **0x0**, 0x100, 0x180], [--, --, --, --]	<2>[0,**0**,**0**,**0**], <0>[0,0,0,0]	---	---	---

Note: way 0 remains in FIFO position for set index 0 for a long time because it is the oldest valid thing inserted into the cache. However, as noted at the top, we always evict invalid entries first. And then later when all ways have valid data, we still don't evict it right away because of its second chance bit.

SRRIP:

Use the following SRRIP replacement algorithm ("hit promotion"/"victim selection" policies) for steady state behavior (i.e., when there aren't any invalid entries):

- **Note**: this is a simplified version of what we'll discuss in class for SRRIP; you'll see why we use this simplified version in Homework #5.
- Cache Hit:
    1. Set counter value of the block that hits to **max**(previous counter value - 1, 0)
- Cache Miss: (**NOTE: SRRIP skips steps 3 and 4 if an invalid entry is found**)
    1. Search for first highest value in this set Way (if multiple entries with the same value, then pick the one with the smaller way number).
    2. Replace block and set counter for newly inserted entry to '2'.
    3. Set: *diff* = 3 - victim's counter value
    4. For all other way's counter values (for this set index):
        1. If (*diff* <= 0) leave all other counter values as is
        2. Else update each of other way's counter values to **min**(way's previous counter value+*diff*, 3).

Cache Contents	SRRIP Vals	Access Address	SRRIP Hit/Miss	Victim (If Any)
[--, --, --, --], [--, --, --, --]	[2, 2, 2, 2], [2, 2, 2, 2]	0x0	Miss	Way 0 (--)
[**0x0**, --, --, --], [--, --, --, --]	[**2**, 2, 2, 2], [2, 2, 2, 2]	0x81	Miss	Way 1 (--)
[0x0, **0x80**, --, --], [--, --, --, --]	[2, **2**, 2, 2], [2, 2, 2, 2]	0x100	Miss	Way 2 (--)
[0x0, 0x80, **0x100**, --], [--, --, --, --]	[2, 2, **2**, 2], [2, 2, 2, 2]	0x188	Miss	Way 3 (--)
[0x0, 0x80, 0x100, **0x180**], [--, --, --, --]	[2, 2, 2, **2**], [2, 2, 2, 2]	0x4	Hit	N/A
[**0x0**, 0x80, 0x100, 0x180], [--, --, --, --]	[**1**, 2, 2, 2], [2, 2, 2, 2]	0x200	Miss	Way 1 (0x80)
[0x0, **0x200**, 0x100, 0x180], [--, --, --, --]	[**2**, **2**, **3**, **3**], [2, 2, 2, 2]	0x18C	Hit	N/A
[0x0, 0x200, 0x100, **0x180**], [--, --, --, --]	[2, 2, 3, **2**], [2, 2, 2, 2]	0x108	Hit	N/A

[0x0, 0x200, **0x100**, 0x180], [--, --, --, --]	[2, 2, **2**, 2], [2, 2, 2, 2]	0x380	Miss	Way 0 (0x0)
[**0x380**, 0x200, 0x100, 0x180], [--, --, --, --]	[**2**, **3**, **3**, **3**], [2, 2, 2, 2]	0x8	Miss	Way 1 (0x200)
[0x380, **0x0**, 0x100, 0x180], [--, --, --, --]	[2, **2**, 3, 3], [2, 2, 2, 2]	---	---	---

0x200 Access: Way 1 is the first with the highest RRPV (counter) value (which is 3) – evict way 1 for 0x200. Diff = 3-3 = 0 → leave other ways' counter values as is.

0x380 Access: Way 0 is the first with the highest RRPV (counter) value (which is 2) – evict way 0 for 0x380. Diff = 3-2 = 1 → increment counters for all other ways in set index 0 by 1.

0x8 Access: Way 1 is the first with the highest RRPV value (which is 3) – evict way 1 for 0x0. Diff = 3-3 = 0 → leave other way's counter values as is.

BRRIP:

For BRRIP use the same replacement algorithm as SRRIP above, **except** set the threshold to be 67% (2/3). This means on insertion into the cache BRRIP does the following instead of step 2 in SRRIP's algorithm:

- Every third insertion (mod 3) gets a *distant* re-reference interval (2^M-1).
- The other two insertions (mod 3) get a *long* re-reference interval (2^M-2).

Note: insertions only affect misses – thus hits do not affect insertion number. So if the pattern is Miss, Miss, Hit, Miss … then the last miss would be Insertion #2 (i.e., insertion 2 mod 3, assuming the first insertion is access 0).

Overall this means the BRRIP algorithm is (**Note**: below code assumes threshold is 67% (2/3) for step 2 of cache miss):

- Cache Hit:
    1. Set counter value of block that hits to **max**(previous counter value - 1, 0)
- Cache Miss: (**NOTE: BRRIP skips steps 3 and 4 if an invalid entry is found**)
    1. Search for first highest value in this set Way (if multiple entries with the same value, then pick the LRU one).
    2. Replace block and set counter for newly inserted entry to:
        - If this insertion mod 3 == 0, set counter value to '3'.
        - Else (i.e., if this insertion mod 3 != 0), set counter value to '2'.
    3. Set diff = 3 – victim counter's value
    4. For all other way's counter values (for this set index):
        - If (*diff* <= 0) leave all other counter values as is
        - Else update each of other way's counter values to **min**(way's previous counter value+*diff*, 3).

Cache Contents	BRRIP Vals	Access Address	BRRIP Hit/Miss	Victim (If Any)

[--, --, --, --], [--, --, --, --]	[3, 2, 2, 3], [2, 2, 3, 2]	0x0	Miss	Way 0 (--)
[**0x0**, --, --, --], [--, --, --, --]	[**3**, 2, 2, 3], [2, 2, 3, 2]	0x81	Miss	Way 1 (--)
[0x0, **0x80**, --, --], [--, --, --, --]	[3, **2**, 2, 3], [2, 2, 3, 2]	0x100	Miss	Way 2 (--)
[0x0, 0x80, **0x100**, --], [--, --, --, --]	[3, 2, **2**, 3], [2, 2, 3, 2]	0x188	Miss	Way 3 (--)
[0x0, 0x80, 0x100, **0x180**], [--, --, --, --]	[3, 2, 2, **3**], [2, 2, 3, 2]	0x4	Hit	N/A
[**0x0**, 0x80, 0x100, 0x180], [--, --, --, --]	[**2**, 2, 2, 3], [2, 2, 3, 2]	0x200	Miss	Way 3 (0x180)
[0x0, 0x80, 0x100, **0x200**], [--, --, --, --]	[2, 2, 2, **2**], [2, 2, 3, 2]	0x18C	Miss	Way 0 (0x0)
[**0x180**, 0x80, 0x100, 0x200], [--, --, --, --]	[**2, 3, 3, 3**], [2, 2, 3, 2]	0x108	Hit	N/A
[0x180, 0x80, **0x100**, 0x200], [--, --, --, --]	[2, 3, **2**, 3], [2, 2, 3, 2]	0x380	Miss	Way 1 (0x80)
[0x180, **0x380**, 0x100, 0x200], [--, --, --, --]	[2, **3**, 2, 3], [2, 2, 3, 2]	0x8	Miss	Way 1 (0x380)
[0x180, **0x0**, 0x100, 0x200], [--, --, --, --]	[2, **2**, 2, 3], [2, 2, 3, 2]	---	---	---

0x200 Access: Way 3 has the highest counter value (which is 3) – evict way 3 for 0x200. Diff = 3-3 = 0 → leave other ways' counter values as is. Moreover this is insertion #4 (4 % 3 != 0) so we insert 0x200 with counter value 2.

0x18C Access: Way 0 has first, highest counter value (which is 2) – evict way 0 for 0x180. Diff = 3-2 = 1 → increment other ways' counter values for set index 0 by 1. Moreover this is insertion #5 (5 % 3 != 0) so we insert 0x180 with counter value 2.

0x380 Access: Way 1 has first, highest counter value (which is 3) – evict way 1 for 0x380. Diff = 3-3 = 0 → leave other ways' counter values as is. Moreover this is insertion #6 (6 % 3 == 0) so we insert 0x380 with counter value 3.

0x8 Access: Way 1 has the first, highest counter value (which is 3) – evict way 1 for 0x0. Diff = 3-3 = 0 → leave other ways' counter values as is. Moreover this is insertion #7 (7 % 3 != 0) so we insert 0x0 with counter value 2.

Solution:

See above. Interestingly, policies like BRRIP struggle because their threshold data attempting to distinguish between different access intervals backfires for this (admittedly arbitrary) set of accesses. In comparison, LFU does really well (among others) because the access pattern here aligns well with its thrash resistant approach.