CS 105 Check-in 3

Gisele Umutoni

TOTAL POINTS

11.5 / 30

QUESTION 1

1 Caching 7.5 / 10

- **0 pts** Correct
- **1 pts** address should have 12-bit tag, 2-bit index, 2-bit offset
- **0.5 pts** i. Hit
- **0.5 pts** i. n/a, No
- **0.5 pts** ii. Hit
- **0.5 pts** ii. 0XAE, No
- **0.5 pts** iii. Miss

✓ **- 0.5 pts** *iii. 0x47, No*

- **0.5 pts** iv. Hit
- **0.5 pts** iv. 0x47, No
- **0.5 pts** v. Miss
- **0 pts** v. n/a, No
- **0.5 pts** vi. Hit

✓ **- 0.5 pts** *vi. 0x47, No*

- **0.5 pts** vii. Miss
- **0.5 pts** vii. 0x47, Yes

✓ **- 0.5 pts** *idx 3, line 0 should be 007 47 01 47 47*

- **0.5 pts** idx 3, line 1 should be 1E0 47 47 47 47

✓ **- 0.5 pts** *idx 3, line 3 should be AAA 47 47 47 47*

✓ **- 0.5 pts** *mistakes with dirty and/or valid bits*

QUESTION 2

2 Optimization with Caches 2 / 10

- **0 pts** Correct

✓ **- 2 pts** *1st loop: miss rate should be 1/2 (50%)*

✓ **- 2 pts** *2nd loop: miss rate should be 1/14 (~7%)*

- **2 pts** 3rd loop: miss rate should be 2/16 (12.5%)

✓ **- 2 pts** *(d) suggestion wouldn't work or doesn't use current hardware*

✓ **- 2 pts** *(e) no, values still wouldn't be in cache when you get to 2nd loop*

QUESTION 3

3 Dynamic Memory 2 / 10

- **0 pts** Correct

✓ **- 2 pts** *did not correctly identify current block to free*

✓ **- 2 pts** *did not correctly coalesce with prev block*

✓ **- 2 pts** *did not correctly coalesce with next block*

✓ **- 2 pts** *header/footer do not have correct size for new block*

- **1 pts** header/footer do not correctly mark allocation status of new block (should be free)
- **1 pts** header/footer do not correctly mark allocation status of next block (should be allocated)

CS 105 Fall 2023
Check-in 3

November 1, 2023

Name: Gissele Umolani

Instructions. There are 3 problems on this check-in. Each problem is worth 10 points. You will have 90 minutes to work on the problems.

This is a closed-book check-in. You may use one 8.5x11" two-sided sheet of notes. Calculators are allowed. But you may not use books, computers, other printed materials, laptops, etc.

Write all of your answers on the checkin itself. **Answers written on scratch paper not be scanned hence will not be graded.**

Data Sizes

`char`	1 bytes
`short`	2
`int`	4
`double`	8
`pointer`	8

Register usage

`%rdi`	first argument
`%rsi`	second argument
`%rdx`	third argument
`%rcx`	fourth argument
`%rax`	function result
`%rsp`	stack pointer

Callee-saved registers

`%rsp`
`%rbp`
`%rbx`
`%r12`
`%r13`
`%r14`
`%r15`

1. Caching [10 pts]

The following table depicts a 4-way set associate, write-back, write-allocate cache, with a 4 byte block size and 16 total lines that uses a least-recently used (LRU) eviction policy.:

4-way Set Associative Cache												
idx	Tag	V	D	Datablock	Tag	V	D	Datablock	Tag	V	D	Datablock	Tag	V	D	Datablock
0	029	0	0	34 29 34 29	787	0	0	39 ae 39 ae	C7D	1	0	68 01 68 f2	88B	1	0	64 38 64 38
1	AF3	1	0	0d 8f ac 1f	C3D	1	0	0c 3a 0c 3a	EB8	1	0	a4 db 0c 3a	6D9	1	0	a5 3c a5 3c
2	2A7	1	0	e2 04 68 01	FAB	1	0	d2 68 01 04	BE3	0	0	3c a4 db a4	D01	1	0	a4 db ee 05
3	23B	1	0	ac 1f 38 64	1E0	0	0	b5 70 ae 39	EB8	1	0	66 39 ae 95	E37	1	0	49 f3 3a ~~0c~~ 05

You should assume: (1) Each memory access reads/writes **1-byte**, (2) The D bit indicates whether that cacheline has been modified, and (3) Any bytes read from memory (i.e., that are not cached) have the value 0x47

(a) [1 pt] The box below depicts a 16-bit memory address. Indicate which bits would be used to determine (1) the tag, (2) the index, and (3) the offset.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1	1	1	1	1	2	2	3	3

(b) [7 pts]

Consider the following sequence of accesses (yes, they occur sequentially). For each access, indicate whether that access would correspond to a cache hit or a cache miss, which byte is read (for reads), and whether or not a memory write will occur.

	Operation	Hit/Miss	Byte read	Mem write?
i.	`Write $0x00, 0xE37F`	Hit	n/a	N
ii.	`Read 0xEB8E`	Hit	ae	N
iii.	`Read 0x1E0D`	Miss	0x47	Y
iv.	`Read 0x1E0C`	Hit	0x47	N
v.	`Write $0x01, 0x007D`	Miss	n/a	N
vi.	`Read 0x007C`	Hit	01	N
vii.	`Read 0xAAAC`	Miss	0x47	Y

(c) [2 pts] Use the diagram below to indicate the state of the cache after the above sequence of accesses completes. You may leave cachelines blank if they don't change.

4-way Set Associative Cache				
idx	Tag V D Datablock	Tag V D Datablock	Tag V D Datablock	Tag V D Datablock
0				
1				
2				
3		1E0 1 1 47 47 47 47		007 1 1 01

2. Optimization with Caches [10 pts] Realizing that next week's election is coming up soon, an electronic voting company asks for your help improving the performance of their software. Their voting software will run on a machine with a 1024-byte direct-mapped data cache with 64 byte blocks. So far, they have the following code for initializing election ballots:

```
struct vote {
    int candidates[7];
    int submitted;
};

struct vote vote_array[16][16];
for (int i=0; i<16; i++){
    for (int j=0; j<16; j++) {
        vote_array[i][j].submitted=0;
    }
}
for (int i=0; i<16; i++){
    for (int j=0; j<16; j++) {
        for (int k=0; k<7; k++) {
            vote_array[i][j].candidates[k] = 0;
        }
    }
}
```

You can assume: (1) `vote_array` begins at memory address `0x602c40`, (2) the cache is initially empty, and (3) the only memory accesses are to the entries of the array `vote_array`. Variables `i`, `j` and `k` are stored in registers.

(a) [2 pts] What is the miss rate for memory accesses in the first loop: 0.0625

(b) [2 pts] What is the miss rate for memory accesses in the second loop: 0.0625

(c) [2 pts] What is the overall miss rate this program: 0.125

(d) [2 pts] Suggest one way to modify this code that would improve performance on the current hardware:

- Separating accumulators

(e) [2 pts] Briefly explain whether running on a machine with a 2-way set-associative cache would improve performance of the current code and why:

running on a machine with a 2-way set-associative cache would improve performance because it would reduce the number of misses

3. Dynamic Memory [10 pts]

Consider a dynamic memory allocator that uses an **implicit** free list. Each memory block, either allocated or free, has a size that is a multiple of four bytes. Thus, only the 30 higher order bits in the header and footer are needed to record block size, which includes the header and footer. In an effort to optimize performance, someone decides to use both "extra" bits to store information about not just the allocation status of the current block but also the allocation status of the next block. More precisely, the usage of the 2 lower order bits is as follows:

- bit 0 (low-order bit) indicates the use of the current block: 1 for allocated, 0 for free.
- bit 1 indicates the use of the next adjacent block: 1 for allocated, 0 for free.

Given the contents of the heap shown on the left, show the new contents of the heap (in the right table) after a call to `free(0x400AFEC)` is executed. Assume that the allocator uses **immediate coalescing**, that is, adjacent free blocks are merged by the `free` function.

Your answers should be given as hex values. Leave a rectangle blank if the values at that address did not change.

Hint: Note that the address grows from bottom up. So the header of each block is at the bottom.

0x400B024	0x00000016
0x400B020	0x00000021
0x400B01C	0x0000001D
0x400B018	0x00000016
0x400B014	0x00000021
0x400B010	0x00000021
0x400B00C	0x0000001E
0x400B008	0x00000016
0x400B004	0x00000021
0x400B000	0x0000001E
0x400AFFC	0x00000021
0x400AFF8	0x00000016
0x400AFF4	0x00000021
0x400AFF0	0x0000001E
0x400AFEC	0x00000016
0x400AFE8	0x00000021
0x400AFE4	0x0000001A
0x400AFE0	0x00000016
0x400AFDC	0x0000001E
0x400AFD8	0x00000021
0x400AFD4	0x0000000D
0x400AFD0	0x0000001A
0x400AFCC	0x0000000D

0x400B024	0x00000036
0x400B020	
0x400B01C	
0x400B018	
0x400B014	
0x400B010	
0x400B00C	
0x400B008	
0x400B004	
0x400B000	
0x400AFFC	
0x400AFF8	
0x400AFF4	
0x400AFF0	
0x400AFEC	
0x400AFE8	
0x400AFE4	
0x400AFE0	
0x400AFDC	
0x400AFD8	0x00000036
0x400AFD4	
0x400AFD0	
0x400AFCC	