CS61C Fall 2016 Guerilla Session #5:
Caches

Q1 Warm up with Tiny Cache

These two warm up questions are a bit long (and may seem tedious), but will hopefully give you a better visualization of caches and a more straightforward way to tackle cache problems.

I have a special cache called Tiny Cache. It is 128 byte direct mapped cache with cache blocks of size 16 bytes, and a physical memory of 4 KiB (1 KiB = 2^10 bytes). I also have some arrays:

```
int A[4] = {0, 1, 2, 3};
int B[8] = {-1, -2, -3, -4, -5, -6, -7, -8};
int C[4] = {1, 2, 4, 8};
```

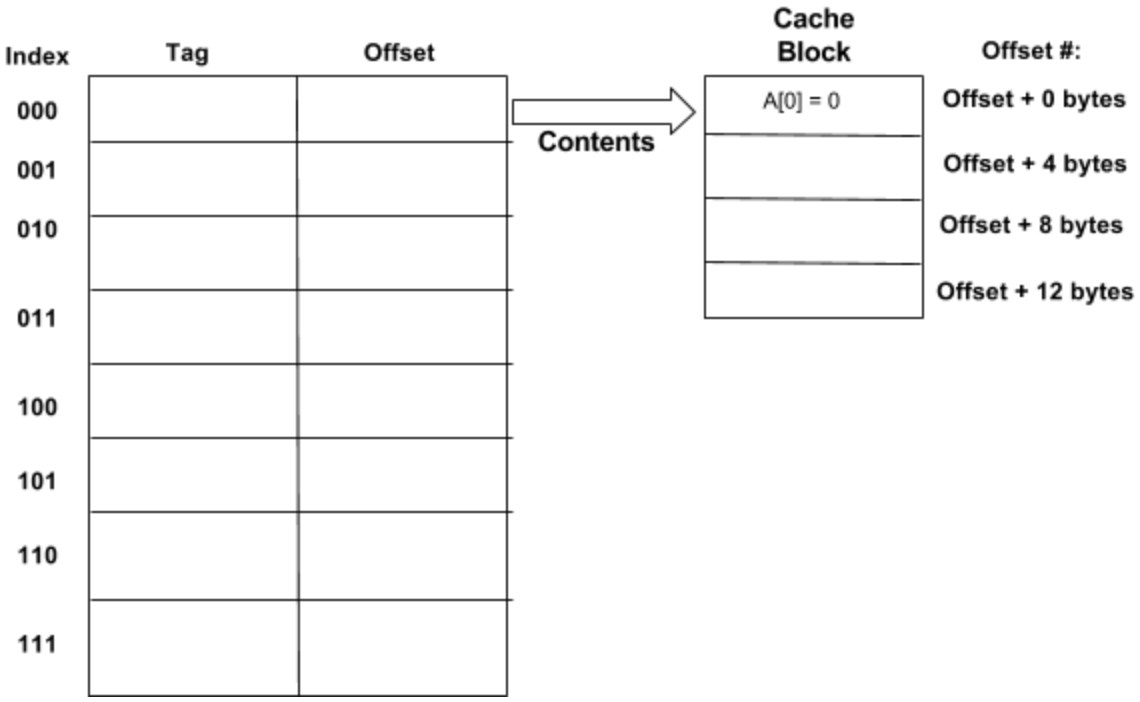
A is located at address 0x100, B is located at 0x310, and C is located at 0x200.

1) *Calculate the sizes of everything*

a. What is the number of tag bits, index bits, and offset bits? How many cache blocks are there in Tiny Cache?
# of offset bits = lg(block size) = lg(16) = **4 bits**
# of entries in cache = total cache size / block size = 128 / 16 = 8 entries/cache blocks
# of index bits = lg(entries) = lg(8) = **3 bits**
# of address bits = lg(total memory) = lg(4 * 2^10) = lg(2^12) = 12 bits
# of tag bits = address bits - index bits - offset bits = 12 - 3 - 4 = **5 bits**

b. Extend this drawing of Tiny Cache (you do not need to redraw the individual cache blocks, just add the number of rows/cache blocks that should be in each row and then fill

in the index column for each row).



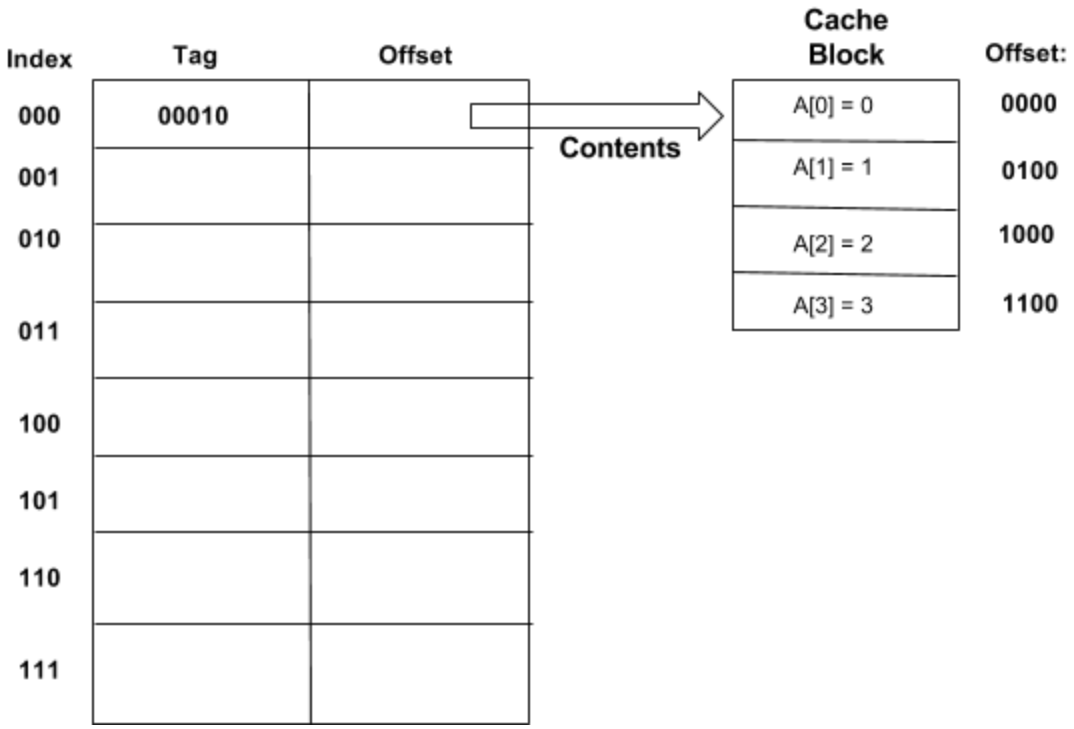
c. How many bytes is an int? How many ints can you store in one cache block?
4 bytes
16 bytes cache block / 4 bytes = 4 ints

d. Can the entire contents of A fit inside a single cache block of Tiny Cache? How about the entire contents of B?
A = yes, B = no (you would need 2 cache blocks)

2) *Figure out the access pattern of the code chunk*
NOTE: All of these code lines are executed sequentially in the same program (i.e. the cache is not cleared after each line).

a. Let's say I execute: `int foo = A[0];` Fill in the drawing from Q1 by labeling which row will now be filled in Tiny Cache, and then filling in the tag column. Then draw an individual cache block and fill out its contents resulting from this code line. It will help to write out the address of A in binary.

e.



b. Next, I execute the for loop below. For each i, do I find A[i] in the cache? What is the hit rate and the miss rate?

```
for (int i = 0; i < 4; i++)
    int x = A[i] + i;
```

Yes I find each A[i] in the cache, therefore, my hit rate = 100% and my miss rate is 0%.

c. Say we changed the body of that for loop to: `A[i] += 4;` How many times is the value A[i] accessed? (i.e. A[i] is read from or written to). Remember that `A[i] += 4` is shorthand for `A[i] = A[i] + 4;`

A[i] is accessed twice. It's first read from once on the right side, and then written to once on the left side.

d. Now, I execute the for loop below. It may be helpful to fill in the drawing from Q1.

```
for (int i = 0; i < 8; i++)
    B[i] += 4
```

i. How many times is the array B accessed? (i.e. how many times is B[i] read from or written to). This is total number of accesses.

B is accessed 16 times.

ii. Is B[i] read from or written to first? (i.e. does the left or right side of the for loop body happen first). This is the first access of B[i] for an iteration of the for loop.

B[i] is read from first.

iii. When i = 0, does the first access of B[i] hit or miss? (i.e. is B[0] in Tiny Cache) How about the second access of B[i]?

It misses, as we have never accessed the array B before, so no data of B is in Tiny Cache (this is called a compulsory miss), and we have to go to main memory and get the value of B[0] (we will also pull B[1], B[2], and B[3] as well to put in our cache). However, the second access hits, as B[0] is now in our cache.

iv. Repeat (ii) for i = 1, 2, 3.

Both the first and second accesses will hit for each of i = 1, 2, 3 as B[1], B[2], and B[3] are all in the cache now after (iii).

v. When i = 4, does the first access of B[i] hit or miss? How about the second access of B[i]? If Tiny Cache instead had block sizes of 32 bytes would your answers change?

The first access will miss and the second access will hit (as in (iii)). If the block size was 32 bytes, then we would have been able to pull B[1] to B[7] as well as B[0] when we first access B[0] in (iii). Therefore, B[4] would have been in Tiny Cache and both accesses would have hit.

vi. Repeat (iv) for i = 5, 6, 7.

After accessing B[4], we will have pulled B[5], B[6], and B[7] from main memory and into Tiny Cache along with B[4]. Therefore the first and second accesses for i = 5, 6, 7 will all hit.

vii. Now you can calculate the total hit rate and miss rate for this for loop. Based on (iii) through (vi), how many accesses hit and how many accesses missed? Divide by the total number of accesses, and what is the total hit rate and miss rate?

2 accesses missed, and 14 accesses missed. Hit rate = 14 accesses / 16 accesses = ⅞. Miss rate = 2 accesses / 16 accesses = ⅛.

e. Next, I execute the line: `int baz = C[0];`

i. Which entry/row of Tiny Cache will the contents of C be entered in? Is that entry/row already occupied? If so, does the occupant have the same or a different tag from C? Draw the contents of the corresponding individual cache block after executing this code line.

The 000 row of Tiny Cache. This row is already occupied by the contents of A. C however has a tag of 00100, while A has a tag of 00010.

C[0] = -1
C[1] = -2
C[2] = -4
C[3] = -8

ii. I then execute the for loop from part (c) again. Will the hit rate and miss rate be different?

It will be different (i.e. A[0]'s first access will miss), as A is no longer in the cache. Therefore the hit rate will be ⅞ while the miss rate will be ⅛.

iii. Let's say Tiny Cache is now a 2-way associative cache of 256 B, with the same block size. Would the behavior of Tiny Cache in (i) have been different? How about the hit and miss rates in (ii)? (Hint: how many cache blocks are there per entry/row now that Tiny Cache is 2-way associative?)

Yes, instead of replacing the contents of A in the cache, we would simply place C's contents in Tiny Cache as well. This is because we would now have two cache blocks in row 000 of Tiny Cache, when it is 2-way associative. Therefore, the hit/miss rates of accessing A again, would have been the same as before in part (c).

2. Which of these instruction dependencies would cause a pipelining hazard?

A. 2 → 3: ori $t3 $t2 0xDEAD → beq $t2 $t3 label

B. **2 → 4:** ori $t3 $t2 0xDEAD → addiu $t2 $t3 6

C. **2 → 5:** ori $t3 $t2 0xDEAD → addiu $v0 $0 10

D. 3 → 4: beq $t2 $t3 label → addiu $t2 $t3 6

E. None of the above

3. If we were given a **branch delay slot**, which instruction would reduce the most amount of pipelining hazards if moved into the branch delay slot? If all instructions are equally beneficial, or no instruction removes any hazards, write "nop" as your answer.

addiu $v0 $0 10

Q4: Cache Rules Everything Around Me (15 points)

You are given a MIPS machine with a single level of **2KiB direct-mapped** cache with **512B cache blocks**. It has **1MiB of physical address space**.

The function foo is ran on the system with a **cold** cache and as the only process:

```
#define ARRAY_LEN 4096
#define STEP_SIZE 64

// A starts at 0x10000
// B starts at 0x20000

foo( int* A, int* B ) {
      int total = 0;
      for ( int i = 0; i < ARRAY_LEN; i += STEP_SIZE ) {
            total += A[ i ];
            total -= B[ i ];
      }
}
```

1. Calculate the number of Tag, Index, and Offset bits for this cache.

Tag: 9

Index: 2

Offset: 9

2. Calculate the hit percentage for this cache after running `foo`.

0%

3. The cache is now cleared and the code is run again. This time, **A** and **B** are pointing to the same array, which starts at **0x10000**. Calculate the new hit percentage.

75%

4. Assume **A** and **B** starts once again at **0x10000** and **0x20000**. What is the new hit percentage if we ran `foo` on a fully associative cache, with all other parameters staying the same?

50%

30 ps	20 ps	30 ps

b. What is the minimum clock period, in picoseconds, for which the processor can run?
260

c. What is the time required, in picoseconds, that it takes for the CPU, starting from the first stage of the lw instruction, to finish the execution of the final sw instruction? You may use the variable stall_cycles in place of the sum of your answers for question a, and clock_period in place of your answer for question b.
(8 + stall_cycles) * clock_period

d. Which timing values, if lowered independently (all other timing remain the same), will allow us to increase the frequency of the CPU? Circle all that apply.

Pipelining Register Clock-to-Q, Pipelining Register Hold time, Pipelining Register Setup time, Instruction Fetch, Register Read, ALU, Memory, Register Write
pipelining register clock-to-q, pipelining register setup time, memory (can be either circled or not circled)

MT2-4: If you do well, it's *clobbering* time! (12 points)

The information for one student in regards to clobbering a single midterm is captured in the data of the following *tightly-packed* struct:

```
typedef struct student {
   int studentID;
   float oldZScore;
   float newZScore;
   int clobber;          /* a value equal to 1 if a student clobbers,
                            0 if otherwise */
} student;
```

We run the following code on a 32-bit machine with a 4 KiB write-back cache. `importStudent()` returns a `struct student` that is in the course roster and that has not been returned by `importStudent()` previously. For simplicity, assume `importStudent()` does not affect the cache.

```
int ARR_SIZE = 512; //Class size rounded down for simplicity
student *61CStudents = (student *) malloc (sizeof(student) * ARR_SIZE);

/* Assume malloc returns a cache block aligned address */
for (int i = 0; i < ARR_SIZE; i++) {                        <=== part I
   61CStudents[i] = importStudent()  <- what does import student do?
}

for (int i = 0; i < ARR_SIZE; i++) {                        <=== part II
   if (61CStudents[i].oldZscore > 61CStudents[i].newZscore){
     61CStudents[i].clobber = 0;
   } else {
```

```
        61CStudents[i].clobber = 1;
      }
    }
```

a. How many bytes is needed to store the information for a single student?
16 bytes

b. Assume that the block size is 32 B. What is the tag:index:offset breakdown of the cache?
20:7:5

c. At the label **`part I`**, assume that `61CStudents` is filled with the correct data. What type of misses will occur from memory accesses during the process? Why?
Compulsory. New Data; 512 * 16 B >= 4 KiB cache

d. Suppose we run the code again and the cache block size is now 8 B long and the cache is direct-mapped. For the for-loop in **part II**, what is the miss rate in the best case scenario (we want the highest hit rate possible)? What type of misses occur?
Capacity, 2/3

e. For the `for`-loop **in part II**, assume that the cache block size is now 128B.

i. If the cache is direct-mapped, what is the hit rate?
8 students per block. 3 Memory accesses per student, 1 miss & 23 hits. 23/24.

ii. If the cache is fully associative, what is the hit rate? Does associativity help? Why or why not?
23/24.

MT2-5: What is the *floating point* of *complex* numbers? (5 points)

We realize that you want to represent complex numbers, which are in the form $a + bi$, where a is the real component, b is the imaginary component, and the magnitude is $\sqrt{a^2 + b^2}$.

We create a 16-bit representation for storing both the real and imaginary components as floating point numbers with the following form: The first 8 bits will represent the real component, and the latter 8 bits will represent the complex component. Our new representation will look like:

Sign	Exponent	Significand	Sign	Exponent	Significand
15	14-12	11-8	7	6-4	3-0

Bits per field:
Sign: 1
Exponent: 3
Significand: 4
Everything else follows the IEEE standard 754 for floating point, except in 16 bits
Bias: 3

SID: _____

Q5) Caches (17 pts)

Assume we are working in a 32-bit physical address space. We have two possible data caches: cache X is a direct-mapped cache, while cache Y is 2-way associative with LRU replacement policy. Both are 4 KiB caches with 512 B blocks and use write-back and write-allocate policies.

a) Calculate the number of bits used for Tag, Index and Offset:

Cache	Tag bits	Index bits	Offset bits
X	20	3	9
Y	21	2	9

Use the code below to answer the following parts. Assume that ints are 4 B and doubles are 8 B.

```
//2^11 elements in the double array
int DOUBLE_ARRAY_SIZE = 2 * 1024;                    //2^14 bytes
double double_arr[DOUBLE_ARRAY_SIZE];

for (int i = 0; i < DOUBLE_ARRAY_SIZE; i++) /* loop 1 */
      double_arr[i] = i;
for (int i = 0; i < DOUBLE_ARRAY_SIZE;  i += 8) /* loop 2 */
     double_arr[i] *= double_arr[0];
```

b) What is the hit rate for each cache if we run only loop 1? (hint: they're both the same). What types of misses do we get?

Both have a hit rate of 63/64. Compulsory

c) What is the hit rate of each cache when you execute loop 2? Assume that you have executed loop 1. Assume the worst case ordering of accesses within a single iteration of the loop if multiple orders are possible. You may leave your answer as an expression involving products and sums of fractions.

X: ____See Below_____ Y: _____See Below_____

We are accessing with a stride of 8*8B = 64B while our block size is 512B. Thus, we have 8 accesses in each block. Since the array size is 2Ki*8B = 16KiB and our caches are 4KiB, the actual data is 4 times the size of our caches.

In cache X, the first quarter of the array will have a hit rate of 23/24 since we only encounter misses in each new block. In the rest of the array, however, we get a ping-pong effect on the first block as the loop requires double_arr[0]. For the 2nd-4th quarter of the array, the first block accesses will be (double_arr[i], double_arr[0], double_arr[i]):
Access 1: M, M, M
Access 2: H, M, M
Access 3: H, M, M; then H, M, M until access 8.
This yields a hit rate of 7/24. The second through eighth block accesses have 23/24 hit rate.

Putting it all together, since we are accessing 4*8 blocks in total,
Hit rate = 29/32 * 23/24 + 3/32 * 7/24

Cache Y has a hit rate of 23/24 because there is no ping-pong effect in the first block.

d) Compute the AMAT for the following system with 3 levels of caches. (You should not need any information from the previous parts of this problem.) Give your answer as a decimal value.

L1$	L2$	L3$	Main Memory
Global miss rate: 50% Hit time: 1ns	Local miss rate: 20% Hit time: 5ns	Local miss rate: 1% Hit time: 15ns	Hit time: 500ns

1 + .5*(5 + .20*(15 + .01*(500))) = 5.5 ns