CS 240 SI Worksheet #19
Valerie Zhao
4/19/17

Exam #2 Review

1. Given an **initially empty** cache with **16-bit addresses** and a **capacity of 512 bytes**, for **byte-addressable** memory:

a. Suppose the cache is **2-way set-associative,** and follows the **write-back** and **least recently used** policies**.** For the following code, if the cache **miss rate is 1/8,** then:

```
char[256][4] A; // A starts at address 0x1000
char[256][4] B; // B starts at address 0x2000
for (i = 0; i < 256; i++) {
     for(j = 0; j < 4; j++) {
          A[i][j] = B[i][j];
     }
}
```

i. How many bytes are in **one block** of the cache?

ii. How many **sets** are in the cache?

iii. How many **bits** are used to encode the **tag** of the blocks?

(Adapted from University of Washington CSE 351 Final Exam - Winter 2017, Problem 1 & 3: https://courses.cs.washington.edu/courses/cse351/17sp/past-exams/cse351-17wi-final.pdf)

b. Suppose the cache is **direct mapped** (but still with a capacity of 512 bytes for byte-addressable memory, using the same policies as in part a).

   i. If **A starts at 0x1000** and **B starts at 0x2000** (like in part a), what is the **miss** rate for the code above if the cache block size is

      1. **4** bytes?

      2. **8** bytes?

      3. **16** bytes?

   ii. If **A starts at 0x1000** and **B starts at 0x1FF0**, what is the **miss** rate for the code above if the cache block size is

      1. **4** bytes?

      2. **8** bytes?

      3. **16** bytes?

2. Consider the following (partially blank) x86-64 assembly, (partially blank) C code, and memory listing. **Addresses and values are 64-bit**, and the machine is **little-endian**. All the values in memory are in hex, and the address of each cell is the sum of the row and column headers: for example, address 0x1019 contains the value 0x18.

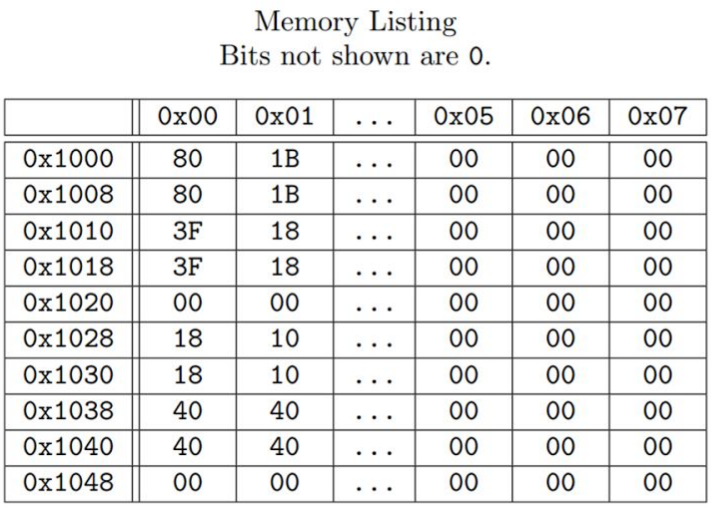
Assembly code:

```
foo:
     movl $0, ______
L1:
     cmpq $0x0, %rdi
     je L2
     cmp _____, 0x1(%rdi)
     je ______
     mov 0x8(%rdi), %rdi
     jmp _______
L2:
     ret
L3:
     mov (%rdi), %eax
     jmp L2
```

C code:

```
typedef struct person {
     char height;
     char age;
     struct person* next_person;
} person;

int foo(person* p) {
     int answer = ______;
     while (_________) {
          if (p->age == 24){
               answer = p-> ________;
               break;
          }
          p = _______________;
     }
     return answer;
}
```



Memory Listing
Bits not shown are 0.

	0x00	0x01	...	0x05	0x06	0x07
0x1000	80	1B	...	00	00	00
0x1008	80	1B	...	00	00	00
0x1010	3F	18	...	00	00	00
0x1018	3F	18	...	00	00	00
0x1020	00	00	...	00	00	00
0x1028	18	10	...	00	00	00
0x1030	18	10	...	00	00	00
0x1038	40	40	...	00	00	00
0x1040	40	40	...	00	00	00
0x1048	00	00	...	00	00	00

a. Complete the assembly and C code.

b. Trace the execution of the call to `foo((person*) 0x1028)` in the following table. Show which instruction is executed in each step until foo returns. In each space, place the assembly instruction and the values of the appropriate registers after that instruction executes. You may leave those spots blank when the value does not change. You might not need all steps listed on the table.

Instruction	**%rdi (hex)**	**%eax (decimal)**
`movl`	`0x1028`	`0`
`cmpq`		
`je`		

c. Briefly describe the value that `foo` returns and how it is computed. Use only variable names from the C version in your answer.