CS 354: Intro to Computer Systems (Spring 2018)

Lecture 13 - Set Associative Caches

The following problem concerns basic cache lookups.

- The memory is byte addressable.
- Memory accesses are to **1-byte words** (not 4-byte words).
- Physical addresses are 12 bits wide.
- The cache is 4-way set associative, with a 2-byte block size and 32 total lines.

In the following tables, **all numbers are given in hexadecimal**. The contents of the cache are as follows:

4-way Set Associative Cache																
Index	Tag	Valid	Byte 0	Byte 1	Tag	Valid	Byte 0	Byte 1	Tag	Valid	Byte 0	Byte 1	Tag	Valid	Byte 0	Byte 1
0	29	0	34	29	87	0	39	AE	7D	1	68	F2	8B	1	64	38
1	F3	1	0D	8F	3D	1	0C	3A	4A	1	A4	DB	D9	1	A5	3C
2	A7	1	E2	04	AB	1	D2	04	E3	0	3C	A4	01	0	EE	05
3	3B	0	AC	1F	E0	0	B5	70	3B	1	66	95	37	1	49	F3
4	80	1	60	35	2B	0	19	57	49	1	8D	0E	00	0	70	AB
5	EA	1	B4	17	CC	1	67	DB	8A	0	DE	AA	18	1	2C	D3
6	1C	0	3F	A4	01	0	3A	C1	F0	0	20	13	7F	1	DF	05
7	0F	0	00	FF	AF	1	B1	5F	99	0	AC	96	3A	1	22	79

Part 1

The box below shows the format of a physical address. Indicate (by labeling the diagram) the fields that would be used to determine the following:

CO The block offset within the cache line
CI The cache index
CT The cache tag

11	10	9	8	7	6	5	4	3	2	1	0

Part 2

For the given physical address, indicate the cache entry accessed and the cache byte value returned **in hex**. Indicate whether a cache miss occurs.

If there is a cache miss, enter "-" for "Cache Byte returned".

Physical address: `0x3B6`

Physical address format (one bit per box)

11	10	9	8	7	6	5	4	3	2	1	0

Physical memory reference

Parameter	Value
Cache Offset (CO)	0x
Cache Index (CI)	0x
Cache Tag (CT)	0x
Cache Hit? (Y/N)	
Cache Byte returned	0x

Part 3

In the 4-way Set Associative Cache given above, list all the hex memory addresses that will **hit** in **Set 7**.

Cache Miss Rate Analysis

You are evaluating the cache performance of the following code on a machine with a **64-byte direct-mapped data cache (C = 64)** with **block size of 16-bytes (B = 16)**.
You are given the following definitions:

```
struct point {
    int x;
    int y;
};
struct point grid[4][4];
int total_x = 0, total_y = 0;
int i, j;
```

You should also assume the following:

- sizeof(int) == 4.
- grid begins at memory address 0.
- The cache is initially empty.
- The only memory accesses are to the entries of the array grid.
- Variables i, j, total_x, and total_y are stored in registers.

A. Determine the cache performance for the following **code snippet 1**:

Code snippet 1:

```
for (i = 0; i < 4; i++) {
    for (j = 0; j < 4; j++) {
        total_x += grid[i][j].x;
    }
}
for (i = 0; i < 4; i++) {
    for (j = 0; j < 4; j++) {
        total_y += grid[i][j].y;
    }
}
```

1. What is the total number of reads that miss in the cache? _____

2. What is the miss rate? _____

B. Determine the cache performance for the following **code snippet 2**:

Code snippet 2:

```
for (i = 0; i < 4; i++) {
    for (j = 0; j < 4; j++) {
        total_x += grid[i][j].x;
        total_y += grid[i][j].y;
    }
}
```

1. What is the total number of reads that miss in the cache? _____

2. What is the miss rate? _____

C. Which of these 2 code snippets is better with respect to cache performance?

a. Code snippet #1

b. Code snippet #2

Why? (just a single line explanation is sufficient)

_____

Linking

Consider the three files **sum.h, sum.c** and **main.c** as shown below and answer the questions that follow.

sum.h

```
=====
#ifndef SUM_H
#define SUM_H
extern int sum(int, int);
#endif
```

sum.c

```
=====
#include "sum.h"
extern int num_ops;
static int global_sum = 0;

int sum(int x, int y)
{
   global_sum += (x+y);
   num_ops++;
   return x+y;
}
```

main.c

```
======
#include "sum.h"
#define SUCCESS 0

int num_ops = 0;

int main()
{
    int a = 10;
    int b = 3;
    int result = sum(a,b);
    return SUCCESS;
}
```

The final executable **`a.out`** is produced by running the following command:

```
% gcc sum.c main.c -m32
```

Questions:

A. The variable `num_ops` is **ONLY declared but not defined** in the file _____

B. The variable `num_ops` is **defined** in the file _____

C. Do the following variables / functions need **relocation** when the executable (a.out) is formed? (Just answer: **Yes** or **No**).

   a. Variable `global_sum` in `sum.c` - _____

   b. Variable `result` in `main.c` - _____

   c. Variable `num_ops` in `main.c` - _____

   d. Function `sum()` in `sum.c` - _____

   e. Function `main()` in `main.c` - _____

D. Can the variable `global_sum` be accessed in the file `main.c` without changing the type of the variable `global_sum` in `sum.c`? Yes OR No.

E. What type of object file is **sum.o** ?
   sum.o is generated using the command: `gcc -c sum.c -m32`

   i. **Relocatable Object File**
   ii. **Executable Object File**

F. Which part of **program memory** (code, data, stack, heap) are the following variables / functions stored?

   a. Variable `global_sum` in `sum.c` - _____

   b. Variable `result` in `main.c` - _____

   c. Variable `num_ops` in `main.c` - _____

   d. Function `sum()` in `sum.c` - _____

   e. Function `main()` in `main.c` - _____