Homework1 – intro to cache

Zewen Hua

1.

(a)

	m	*C*	*B*	*E*	*S*	*t*	*s*	*b*
C1	16	64	4	1	16	10	4	2
C2	16	64	16	1	4	10	2	4

(b)

C1:

ACCESS	LOAD	SET	MISS(M)/HIT(H)
AA00	AA00-AA03	Set0	M
AA04	AA04-AA07	Set1	M
AA08	AA08-AA0B	Set2	M
AA05		Set1	H
AA14	AA14-AA17	Set5	M
AA11	AA10-AA13	Set4	M
AA13		Set4	H
AA38	AA38-AA3B	Set14	M
AA09		Set2	H
AA0B		Set2	H
AA04		Set1	H
AA2B	AA28-AA2B	Set10	M
AA05		Set1	H
AA06		Set1	H
AA09		Set2	H
AA11		Set4	H

Hits and misses are shown in the table above and Set 0, 1, 2, 4, 5, 10, 14 have content, Set 3, 6, 7, 8, 9, 11, 12, 13, 15 are empty.

C2:

ACCESS	LOAD	SET	MISS(M)/HIT(H)
AA00	AA00-AA0F	Set0	M
AA04		Set0	H
AA08		Set0	H
AA05		Set0	H
AA14	AA10-AA1F	Set1	M
AA11		Set1	H
AA13		Set1	H
AA38	AA30-AA3F	Set3	M
AA09		Set0	H
AA0B		Set0	H
AA04		Set0	H
AA2B	AA20-AA2F	Set2	M
AA05		Set0	H
AA06		Set0	H
AA09		Set0	H
AA11		Set1	H

Hits and misses are shown in the table above and Set 0, 1, 2, 3 all have content.

(c)

AA00, AA44, AA01, AA45, AA02, AA46, AA03, AA47

For C1: 2 “Miss”, 6 “Hit”

For C2: 8 “Miss”, 0 “Hit”

2.

(a)

Address space m = t + s + b = 22 + 8 + 4 = 34

(b)

Usable cache size C = B $\times$ E $\times$ S

B = 2^b = 16; E = 1; S = 2^s = 256

C = 16 $\times$ 1 $\times$ 256 = 4096

(c)

Apart from “usable cache size,” we still need some bits for “valid bit” and “tag bits” to implement the cache.

(d)

num = 2^t = 2^{22} = 4194304

3.

given:

	m	*C*	*B*	*E*	*S*	*t*	*s*	*b*
C1	64	2^{20}	1024	1	1024	44	10	10

·let sizeof(int) = 4 bytes

·let @array [1000000] = AAAAAAAA00000000

·assume **mini** in registers

Code:

```
int minimum(){
        mini = array[0];
        for(i = 1;i < 1000000;i++){
                if(mini > array[i])
                        mini = array[i];
        }
        return mini;
}
```

4.
given array[10][10];
first case:

```
for(i=0;i<10;i++){
        for(j=0;j<10;j++){
                sum+=array[i][j];
        }
}
average = sum/100;
```

second case:

```
for(j=0;j<10;j++){
        for(i=0;i<10;i++){
                sum+=array[i][j];
        }
}
average = sum/100;
```

In the first case, I add numbers row by row. And in the second case, I add numbers column by column.

The first case has a better locality of reference. Because C is row-major, so the locality of reference for adding row by row can be better.

But in Fortran, it is column major. So, the locality of reference of the second case will be better.

5.

As the variables given, b=4, s=8. So, the address starts like this:

1000 1000 0000 0000 0010 / 0000 0000 / 1000

LOAD	SET	RAM access
ValueA[0-1]	Set0	1
ValueA[2-5]	Set1	2
ValueA[6-9]	Set2	3
…	…	…
ValueA[1018-1021]	Set255	256
ValueA[1022-1025]	Set0	257
ValueA[1026-1029]	Set1	258
…	…	…
ValueA[2042-2045]	Set255	512
ValueA[2046-2047]	Set0	513

After the completion of extended loop, the content that remains in the cache shows here:

SET	VALUE
Set0	ValueA[2046-2047] & ValueA[1024-1025]
Set1	ValueA[1026-1029]
Set2	ValueA[1030-1033]
…	…
Set255	ValueA[2042-2045]

As the first table shows, there are 513 RAM accesses performed.