NWEN242 LAB 3

Vincent Yu —— 300390526

October 23, 2017

1 4.1

a

R2 (v0) = 100004a0
R28 (gp) = 10008000
R5 (a1) = 00000002
R6 (a2) = 0000000c
R29 (sp) = 7fffe858

b

Set	V	LRU	Tag	Data Way0	Acc
0	1	0	400012	00000001 00000001 00000001 00000001	
1	1	0	400012	00000002 00000002 00000002 00000002	hit

DATA cache : Accesses: 8 Hits: 6 Hit Rate : 0.75000
As the table shown above, we can find the value in $Array_A$ all store in the cache.
The cache we have is a 4 ways-associative and each block has the size of 16B.
The cache hold 1,1,1,1 of array A in one block and hold 2,2,2,2 in an other block. In order to know how this works we need to convert 1 and 2 to binary numbers. 1=0000 0001 2=0000 0010 The index of set hold 1 is 0. The index of the set hold value" 2 " is set"1".
The index determined which block need to go so it can mapped the index with sets. And the offset determined which way need to go. If there is a hit, then it will push the whole block up to the processor.

c

Set	V	Tag	Instruction	Acc
0	1	8000	lui $1, 4096	
1	1	8000	ori $2, $1, 1152	
2	1	8000	lui $1,4096	
3	1	8000	ori $8, $1, 1184	
4	1	8000	ori $6, $0, 0	
5	1	8000	ori $4, $0, 8	
6	1	8000	lw $5, 0($2)	
7	1	8000	lw $7, 0($8)	
8	1	8000	add $6, $6, $5	
9	1	8000	add $6, $6, $7	
10	1	8000	addi $2, $2, 4	
11	1	8000	addi $8, $8, 4	
12	1	8000	addi $4, $4, -1	
13	1	8000	slt $1, $0, $4	
14	1	8000	bne $1, $0, -32	
15	1	8000	NULL	

Because of display temporal locality, the most referenced instructions are hold in cache.

d

Set	V	LRU	Tag	Data Way0	Acc
0	1	0	400012	00000001 00000001 00000001 00000001	
1	1	0	400012	00000002 00000002 00000002 00000002	
2	1	0	400012	00000003 00000003 00000003 00000003	
3	1	0	400012	00000004 00000004 00000004 00000004	hit

The idea of mapping is the same with b. But as shown above we have more sets hold value which means we will use more space in the cache to store all the values.

4.2

a

Block size 16B: Hit rate:0.75
Block Size 8B : Hit rate:0.5
Block size 4B : Hit rate:0
Decreasing the block size, there is less data hold in every blocks. This will decrease the hit rate. For example if we put number 1 into the cache it automatically filled cache with numbers 1,2,3,4 (in binary version). If next instruction ask for 2 it will be a hit. But if we decrease the block size. And it only can hold one number in the block. The next instruction also want number 2. But this time it will be a miss, because the block doesn't hold the value.

b

N=1: Value in Register 6(sum result) :28(hexadecimal) 40(decimal) Hit Rate:0.75
N=5: Value in Register 6(sum result) :c8(hexadecimal) 200(decimal) Hit Rate:0.95
N=10: Value in Register 6(sum result) :190(hexadecimal) 400(decimal) Hit Rate:0.975
N=100: Value in Register 6(sum result) :fa0(hexadecimal) 4000(decimal) Hit Rate:0.9975
N=1000: Value in Register 6(sum result) :9c40(hexadecimal) 40000(decimal) Hit Rate:0.99975
As N increasing the hit rate also increasing. I think this is because of the replace policy and temporal locality. We increase the iterations, it will increase the temporal locality. This will increase the hit rate.

4.3

a

Set	V	LRU	Tag	Instruction	Acc
0	1	1	100001	00000041 00000042 000000043	

The data cache store the values. But the cache we have it can only stores 64 numbers in the cache. So it follows LRU which means less recently used rule to replace . The data begins to overwrite the previous data once it is full. Because we are using direct mapping so it just start reimplementing from the beginning.

b

Way	Set	V	FIFO	Tag	Instruction	Acc
0	0	1	1	400006	00000060 00000061 000000062 00000063	
1	0	1	0	400007	00000070 00000071 000000072 00000073	

As we decrease the cache size, it will increase the time of overwriting the data in the cache.
The first data entry into set 0 of the data cache is 60 which indicates the values have been overwritten 3 times.

c

Way	Set	V	FIFO	Tag	Instruction	Acc
0	0	1	1	400006	00000060 00000061 000000062 00000063	
1	0	1	0	400007	00000070 00000071 000000072 00000073	

FIFO is same with LRU. Because we use all the value in the cache , so the result is the same. If we only use a part of data in the cache. FIFO will replace the data from beginning, but LRU will replace the one has been used.

4.4

N	Stride	Hit rate	Misses	Hit rate(instruction)	misses
1	1	0.75	32	0.980695	20
1	2	0.5	32	0.961832	20
1	4	0	32	0.925373	20
1	8	0	16	0.857143	20
1	16	0	8	0.736842	20
1	32	0	4	0.545455	20
1	64	0	2	0.285714	20
5	1	0.75	32	0.996118	20
5	2	0.5	32	0.992284	20
5	4	0	32	0.984756	20
5	8	0	16	0.970238	20
5	16	0	8	0.943182	20
5	32	0.8	4	0.895833	20
5	64	0.8	2	0.895833	20
10	1	0.75	32	0.998	20
10	2	0.5	32	0.996	20
10	4	0	32	0.992	20
10	8	0	16	0.97	20
10	16	0	8	0.97	20
10	32	0.9	4	0.94	20
10	64	0.9	2	0.90	20
100	1	0.75	32	0.999	20
100	2	0.5	32	0.999	20
100	4	0	32	0.999	20
100	8	0	16	0.998	20
100	16	0	8	0.997	20
100	32	0.99	4	0.994	20
100	64	0.99	2	0.990	20

Changing N can change the numbers of iterations. Changing stride value can change the step of using the numbers. For example stride is one then it will sum all the numbers up, but if stride is 4 it will only add 0 + 4 + 8....+ 127 Because $8 represent N and stride is the changing the step in the for loop.

4.5

Mapping type	Block Size	Replacement	Hit Rate
Direct Mapping	4B	LRU	0.666667
Direct Mapping	8B	LRU	0.666667
Direct Mapping	16B	LRU	0.833333
Direct Mapping	4B	FIFO	0.666667
Direct Mapping	8B	FIFO	0.666667
Direct Mapping	16B	FIFO	0.833333
2 Way	4B	LRU	0.666667
2 Way	8B	LRU	0.666667
2 Way	16B	LRU	0.833333
2 Way	4B	FIFO	0.666667
2 Way	8B	FIFO	0.666667
2 Way	16B	FIFO	0.833333
4 Way	4B	LRU	0.666667
4 Way	8B	LRU	0.666667
4 Way	16B	LRU	0.833333
4 Way	4B	FIFO	0.666667
4 Way	8B	FIFO	0.666667
4 Way	16B	FIFO	0.833333
Fully	4B	LRU	0.666667
Fully	8B	LRU	0.666667
Fully	16B	LRU	0.833333
Fully	4B	FIFO	0.666667
Fully	8B	FIFO	0.666667
Fully	16B	FIFO	0.833333