Lund University

Computer Architecture, EITF20

Exercise 2 Solutions

Mohammad Attari (Masoud)

December 17, 2019

Problem 1

In the following `i1` and `i2` are instructions immediately after the branch.

a)

Instruction	Clock cycle no.																		
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
ld x1, 0(x2)	F	D	X	M	W														
addi x1, x1, 1		F	D	D	D	X	M	W											
sd x1, 0(x2)			F	F	F	D	D	D	X	M	W								
addi x2, x2, 4						F	F	F	D	X	M	W							
sub x4, x3, x2									F	D	D	D	X	M	W				
bnez x4, Loop										F	F	F	D	D	D	X	M	W	
i1													F	F	F	D			
i2																F			
ld x1, 0(x2)																	F	D	X

There are 16 cycles between loop instances, the last loop takes 18 cycles:

X3 is x2+40 so the loop takes 10 iterations. 9 loops with 16 cycles, and 1 with 18. T=9*16+18

b)

Instruction	Clock cycle no.																		
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
ld x1, 0(x2)	F	D	X	M	W														
addi x1, x1, 1		F	D	D	X	M	W												
sd x1, 0(x2)			F	F	D	X	M	W											
addi x2, x2, 4					F	D	X	M	W										
sub x4, x3, x2						F	D	X	M	W									
bnez x4, Loop							F	D	D	X	M	W							
i1								F	F										
ld x1, 0(x2)										F	D	X	M	W					

There are 9 cycles between loop instances, the last loop takes 12 cycles:

X3 is x2+40 so the loop takes 10 iterations. 9 loops with 9 cycles, and 1 with 12. T=9*9+12

c)

Instruction	Clock cycle no.																		
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
ld x1, 0(x2)	F	D	X	M	W														
addi x1, x1, 1		F	D	D	X	M	W												
sd x1, 0(x2)			F	F	D	X	M	W											
addi x2, x2, 4					F	D	X	M	W										
sub x4, x3, x2						F	D	X	M	W									
bnez x4, Loop							F	D	X	M	W								
i1								F	D										
ld x1, 0(x2)									F	D	X	M	W						

There are 8 cycles between loop instances, the last loop takes 12 cycles:

X3 is x2+40 so the loop takes 10 iterations. 9 loops with 8 cycles, and 1 with 12. T=9*8+11

Problem 2

a) States: F = Fetch, D = Decode, E = Execute, M = Memory, W = Writeback

Instruction	Clock cycle no.																		
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
LW R5, (R3)	F	D	E	M	W														
LW R6, 8(R3)		F	D	E	M	W													
MUL R5, R5, R6			F	D	D	D	E1	E2	E3	E4	E5	M	W						
ADD R3, R3, R2				F	F	F	D	D	D	D	D	E1	E2	M	W				
SW R5, (R3)					-	-	F	F	F	F	F	D	D	D	D	E	M	W	

b) States: I = Issue, E = Execute, M = Memory, W = Writeback

Instruction	Clock cycle no.																		
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
LW R5, (R3)	I	E	M	W															
LW R6, 8(R3)		I	E	M	W														
MUL R5, R5, R6			I	-	-	E1	E2	E3	E4	E5	W*								
ADD R3, R3, R2				I	E1	E2	W*												
SW R5, (R3)					I	-	-	E**	-	-	-	M	W						

*No M stage needed for Mul and Add

**SW can start E in cycle 8 since it's only address calculation and by that cycle R3 is already read from CDB

c) States: I = Issue, R=ReadOp, E = Execute/Memory, W = Writeback

Instruction	Clock cycle no.																		
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
LW R5, (R3)	I	R	E	W															
LW R6, 8(R3)		I	R	E	W														
MUL R5, R5, R6					I*	R	E1	E2	E3	E4	E5	W							
ADD R3, R3, R2						I	R	E1	E2	W									
SW R5, (R3)							I	-	-	-	-	-	R	E	W				

*Issue has to wait for WAW hazard on R5

problem 3

$$\text{miss rate} = \frac{\text{misses}}{\text{memory accesses}} = \frac{\text{misses / instruction}}{\text{memory accesses / instruction}}$$

$$\text{miss rate}_{16KB\ inst} = \frac{3.82/1000}{1} = .004$$

$$\text{miss rate}_{16KB\ data} = \frac{40.9/1000}{0.36} = .114$$

$$\text{miss rate}_{32KB\ unified} = \frac{43.3/1000}{1 + 0.36} \simeq .0318 = 3.18\%$$

$$\%\ \text{instruction references} = \frac{1}{1.36} = 74\%$$

$$\%\ \text{data references} = \frac{.36}{1.36} = 26\%$$

$$\text{miss rate}_{split} = (74\% \times .004) + (26\% \times .114) = .0326 = 3.26\%$$

$$3.18\% < 3.26\%$$

$\Rightarrow$ unified cache comes out on top by a bit

when we use effective miss rate as a performance metric

problem 3

Average memory access time (AMAT) =

% instruction references × (hit time + instruction miss rate × miss penalty)

+ % data references × (hit time + data miss rate × miss penalty)

$AMAT_{split} = 74\% \times (1 + .004 \times 200) + 26\% (1 + .114 \times 200)$

$= 7.52$

alternatively:

$AMAT_{split} = 1 + 3.26\% \times 200 = 7.52$

extra hit cycle

$AMAT_{unified} = 74\% (1 + .0318 \times 200) + 26\% (1 + 1 + .0318 \times 200)$

$= 7.62$

alternatively:

$100\% (1 + .0318 \times 200) + 26\% (1) = 7.62$

$7.52 < 7.62$

split cache has a better AMAT

Problem 4

$$\text{execution time} = IC \times \left(CPI_{ideal} + \frac{\text{mem accesses}}{\text{instruction}} \times \text{miss rate} \times \text{miss penalty} \right) \times T_C$$

$$\text{speedup} = \frac{1}{1 - p + \frac{p}{s}}$$

$$\text{execution time}_{\text{enhanced}} = IC \times \left(\frac{CPI_{ideal}}{\text{speedup}} + \quad \right) \times T_C$$

1) $$\text{speedup}_{int} = \frac{1}{1 - .423 + \frac{.423}{1.8}}$$

$$\text{execution time}_{INT} = IC \times \Big(2.3 \left(1 - .423 + \frac{.423}{1.8} \right) + \underbrace{\left(1 + .2375 + .0966 \right)}_{1.3341} \times 0.207784 \times 50 \Big) \times TC$$

(1 → instructions; .2375 + .0966 → load/store)

$$= 15.72$$

2) $$\text{speedup}_{data} = \frac{1}{1 - (.2375 + .0966) + \frac{(.2375 + .0966)}{2}}$$

$$\text{execution time}_{data} = IC \times \left(2.3 \times \left(1 - .3341 + \frac{.3341}{2} \right) + 1.3341 \times 0.207789 \times 50 \right) \times T_C$$

$$= 15.78$$

problem 4

c)

$$\text{Execution time}_{\text{cache}} = IC \times (2.3 + 1.3341 \times 0.136385 \times 50) \times 1.1 \times T_C$$

$$= 12.59$$

→ best performance

Problem 5

a)

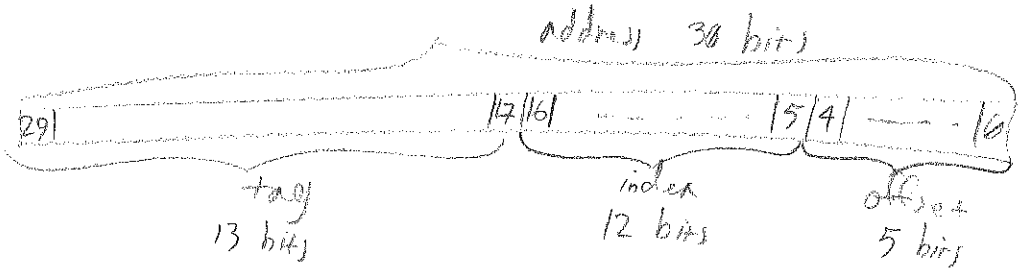
1GB memory ⟶ $1GB = 2^{30}$ ⟶ 30-bit address

$$\text{no of sets} = 2^{\text{index}} = \frac{\text{cache size}}{\text{block size} * \text{blocks/set}}$$

$$= \frac{512KB}{32B * 4} = \frac{2^{19}}{2^5 * 2^2}$$

$2^{\text{index}} = 2^{12}$ ⟶ index = 12 bits

block size = $32B = 2^5 B$ ⟶ offset = 5 bits



tag = 30 − 12 − 5 = 13

problem 5

b) ① break down the address into fields

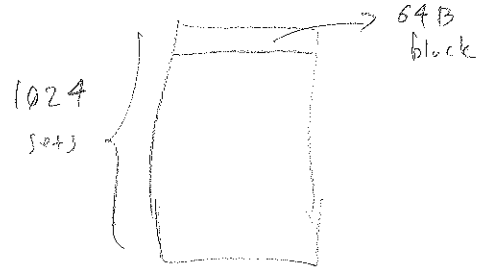
② select the set based on index

③ read the tags (2, since it's 2-way associative) in the cache
↳ Compare with the address tag (valid bit must be set)

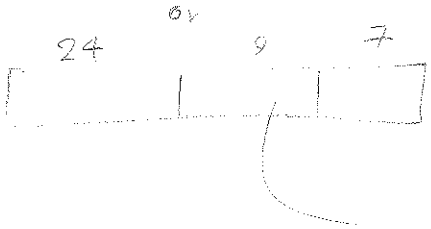
④ if it was a hit, send proper portion to the cpu,
if it was a miss, get the block from (some level memory)

c)

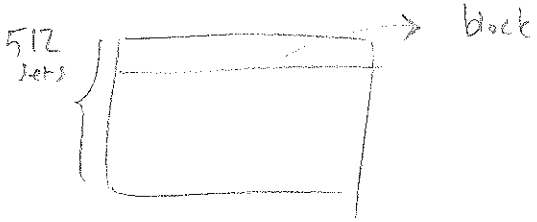
→ 1024 sets
Same block size



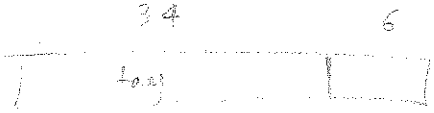
Direct-mapped:



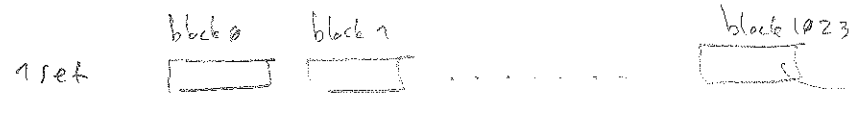
→ 512 sets
double sized blocks



Fully associative:

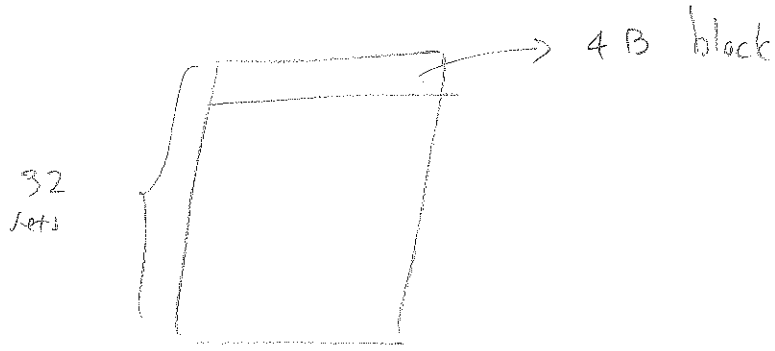


Same cache size means 1024 blocks in 1 set!



problem 5

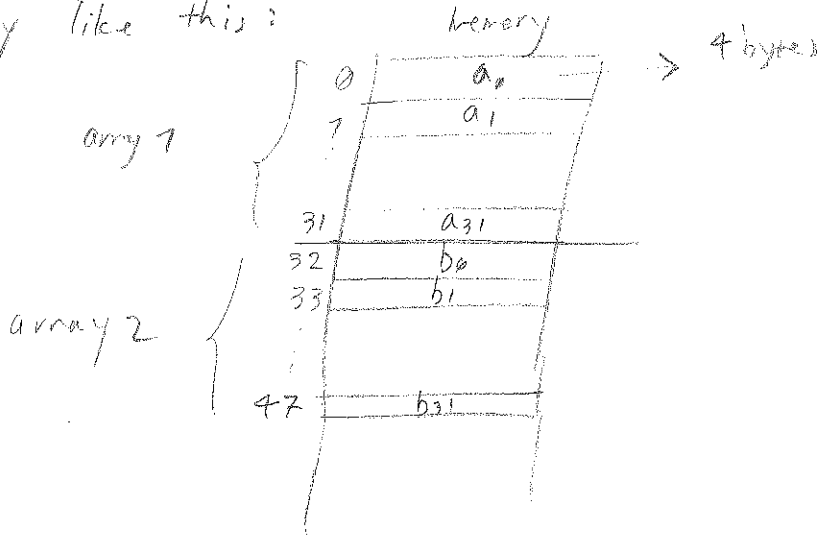
d) the cache looks like this:



the loop iterates over 2 arrays (pointed to by R5 and R6)

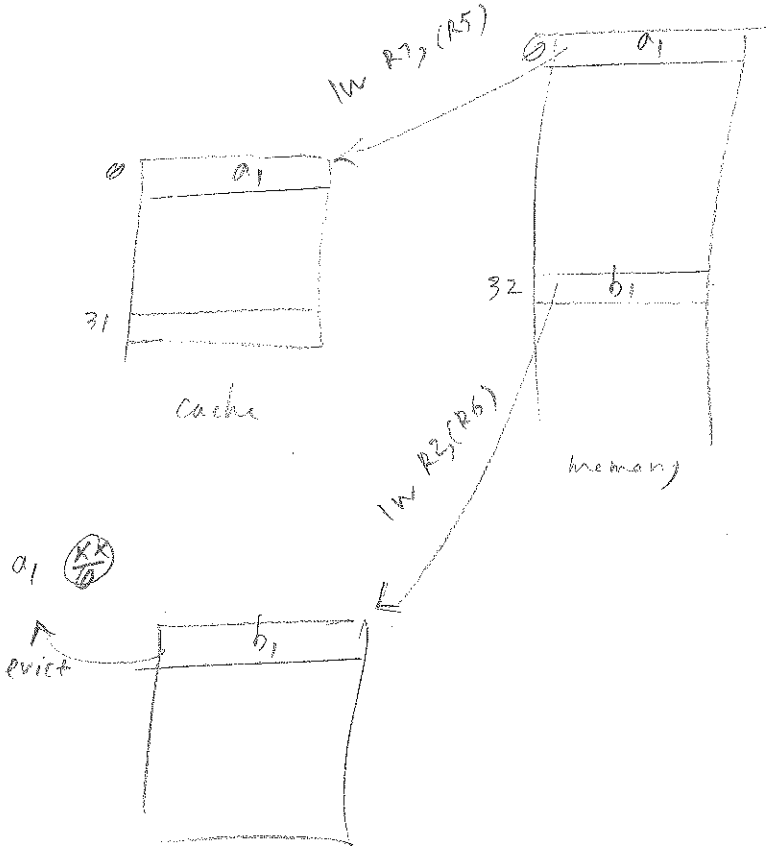
for 32 iterations, loads one of the arrays' elements in each iteration, adds them up, stores the sum in R4, and repeats for the next element in the arrays.

with no block size changes, and assuming the arrays are stored in main memory like this:

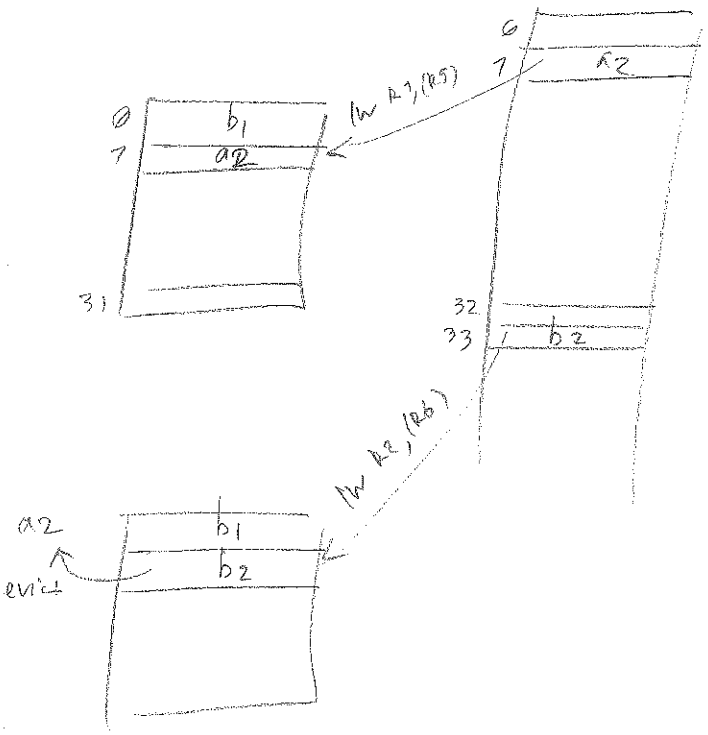


each LW brings 4B (a block) into the cache.

so in the first iteration of the loop, we would have:



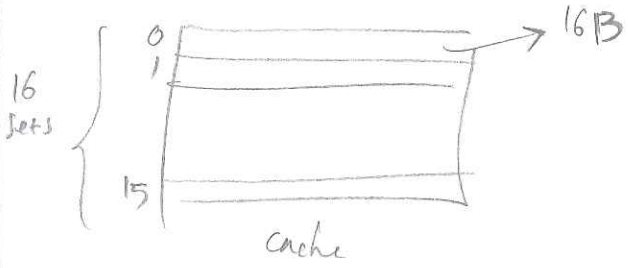
2nd iteration :



and so on!

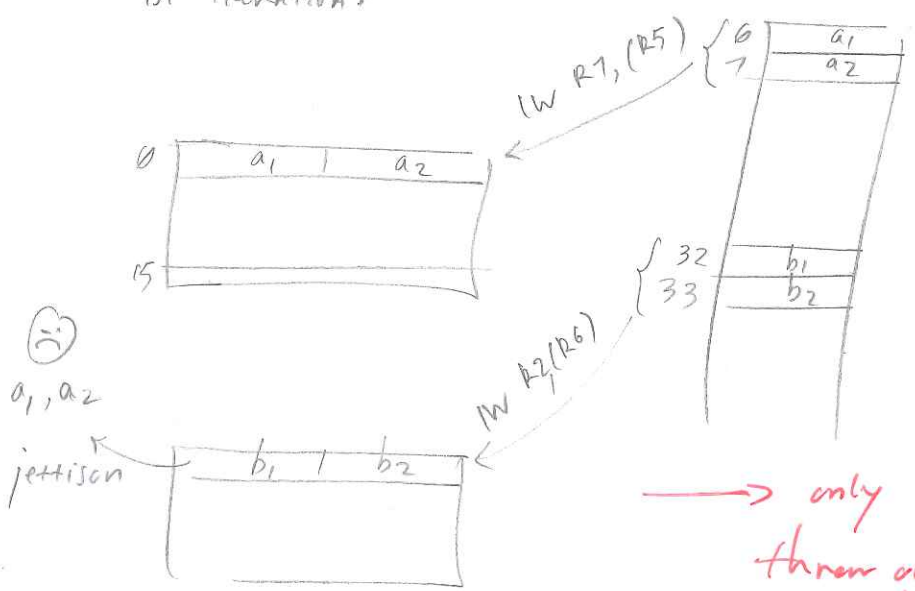
problem 5

now by increasing block size to 8B & reducing sets to 16 we have:



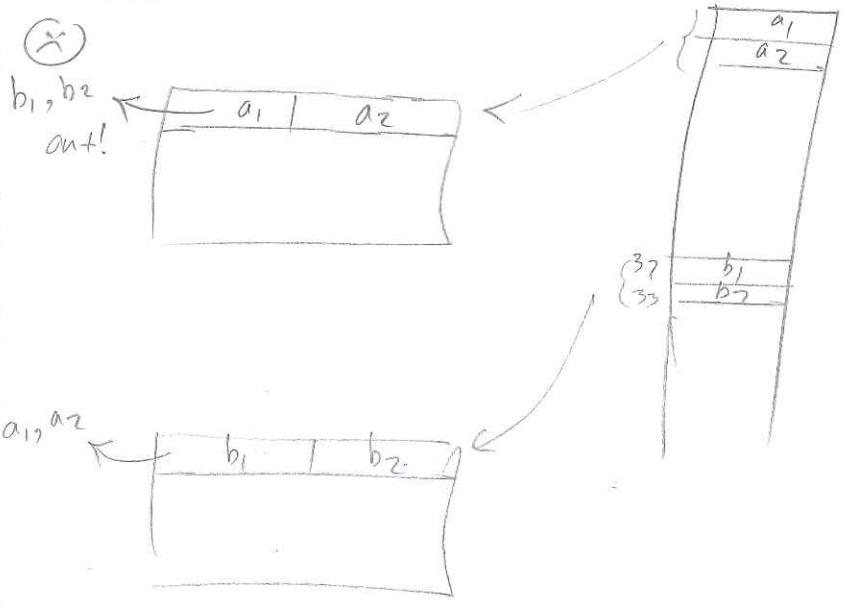
now the loop if arrays stay the same in memory

1st iteration:



→ only consumed a_1, had to throw out a_2 because b_1 & b_2 were brought in

2nd iteration:



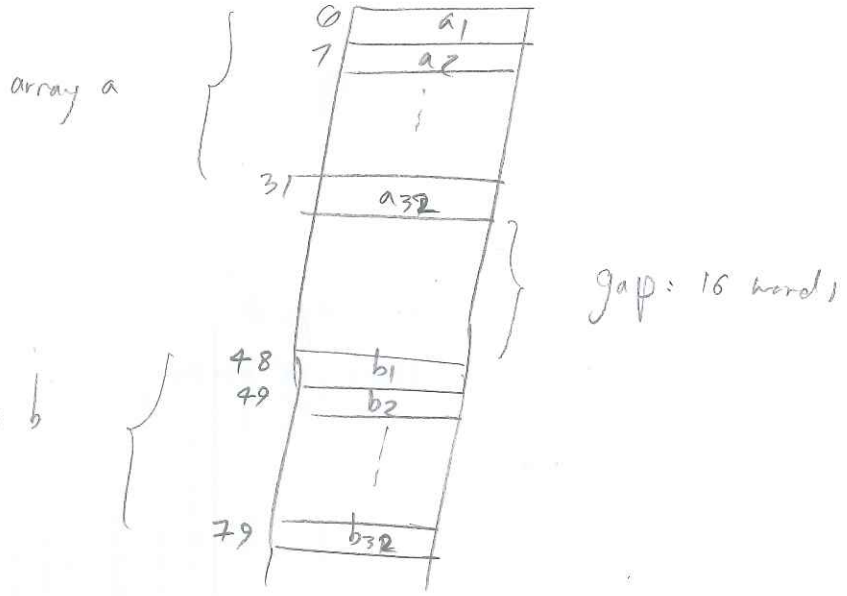
→ this read throws out b_2, before it is utilized

have to load it again!

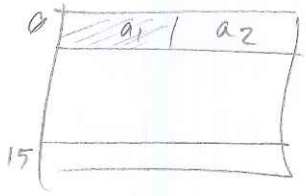
& ...

now if

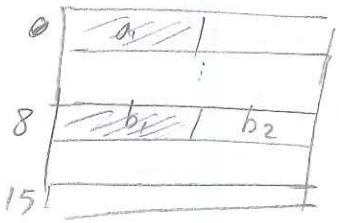
the arrays are placed like this in memory:



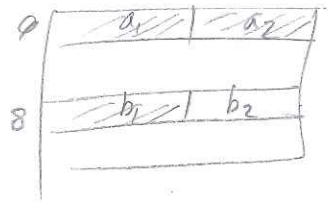
Cache after 1st load, 1st iteration:



Cache after 2nd load, 1st iteration:

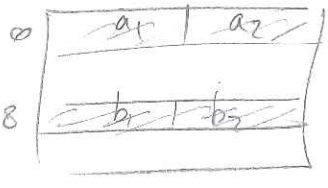


cache after 1st load, 2nd iteration



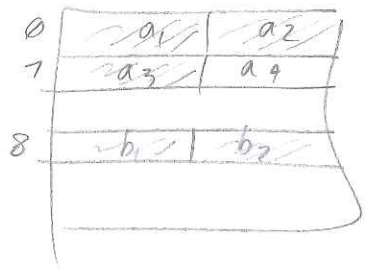
→ a_2 already in the cache. yea baby! ☺

cache after 2nd load, 2nd iteration



→ b_2 also in the cache. yay!

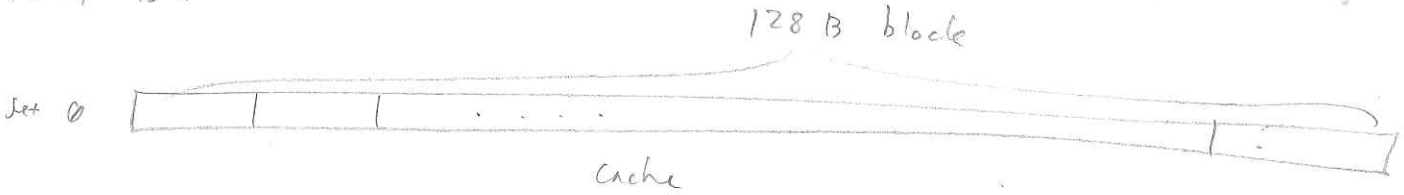
cach after 1st load, 3rd iteration



& . . .

---

in the extreme case, we have a block size of 128 B, just 1 set! that is:



1st load, 1st iteration:

brought 32 elements, consumed only 1.

2nd load, 1st iteration:

each time we bring 32 elements & consume only one & throw the rest out.