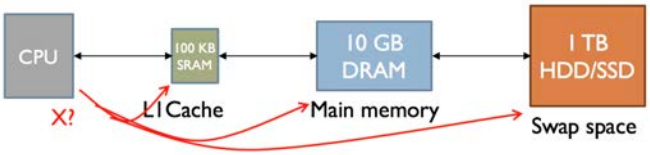
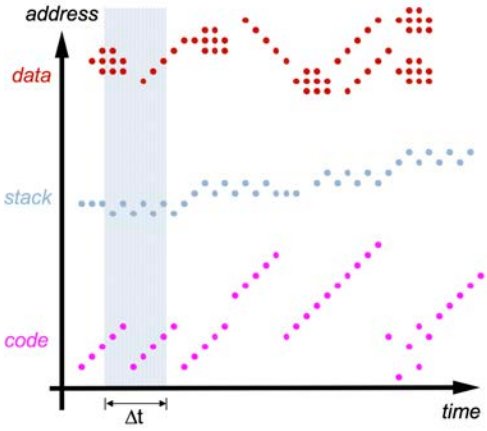
Computation Structures

Memory Hierarchy & Caches Worksheet

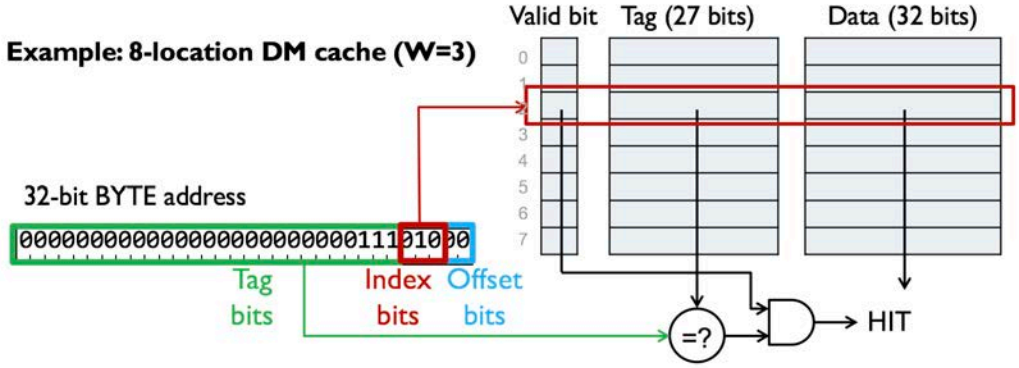




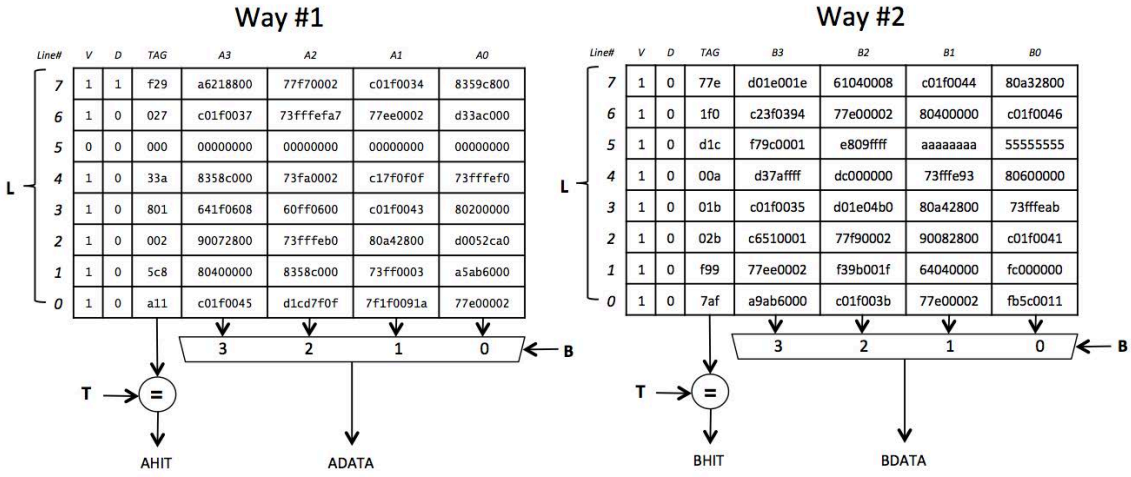
Keep the most often-used data in a small, fast SRAM (often local to CPU chip). The reason this strategy works: LOCALITY.

Locality of reference: Access to address X at time t implies that access to address X+ΔX at time t+Δt becomes more probable as ΔX and Δt approach zero.

AMAT = HitTime + MissRatio * MissPenalty



Example: 2-way set-associative cache, 8 sets, 4-word block size, write-back



Line#	V	D	TAG	A3	A2	A1	A0
7	1	1	f29	a6218800	77f70002	c01f0034	8359c800
6	1	0	027	c01f0037	73fffefa7	77ee0002	d33ac000
5	0	0	000	00000000	00000000	00000000	00000000
4	1	0	33a	8358c000	73fa0002	c17f0f0f	73fffef0
3	1	0	801	641f0608	60ff0600	c01f0043	80200000
2	1	0	002	90072800	73fffeb0	80a42800	d0052ca0
1	1	0	5c8	80400000	8358c000	73ff0003	a5ab6000
0	1	0	a11	c01f0045	d1cd7f0f	7f1f0091a	77e00002

Line#	V	D	TAG	B3	B2	B1	B0
7	1	0	77e	d01e001e	61040008	c01f0044	80a32800
6	1	0	1f0	c23f0394	77e00002	80400000	c01f0046
5	1	0	d1c	f79c0001	e809ffff	aaaaaaaa	55555555
4	1	0	00a	d37affff	dc000000	73fffe93	80600000
3	1	0	01b	c01f0035	d01e04b0	80a42800	73fffeab
2	1	0	02b	c6510001	77f90002	90082800	c01f0041
1	1	0	f99	77ee0002	f39b001f	64040000	fc000000
0	1	0	7af	a9ab6000	c01f003b	77e00002	fb5c0011

Replacement strategy choices: least-recently used (LRU); first in, first out (FIFO); random
Write-policy choices: write-through, write-behind, write-back

Problem 1.

(A) The timing for a particular cache is as follows: checking the cache takes 1 cycle. If there's a hit the data is returned to the CPU at the end of the first cycle. If there's a miss, it takes 10 *additional* cycles to retrieve the word from main memory, store it in the cache, and return it to the CPU. If we want an average memory access time of 1.4 cycles, what is the minimum possible value for the cache's hit ratio?

$1.4 = 1 + (1-\alpha)10$

$\therefore 0.4 = 1-\alpha$

Minimum possible value of hit ratio: 0.96

(B) If the cache block size, i.e., words/cache line, is doubled but the total number of data words in the cache is unchanged, how will the following cache parameters change? Please circle the best answer.

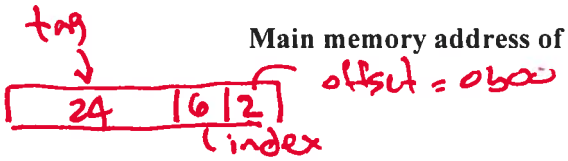
# of offset bits: UNCHANGED ... **+1** ... -1 ... 2x ... 0.5x ... CAN'T TELL

# of tag bits: **UNCHANGED** ... +1 ... -1 ... 2x ... 0.5x ... CAN'T TELL

# of cache lines: UNCHANGED ... +1 ... -1 ... 2x ... **0.5x** ... CAN'T TELL

Consider a direct-mapped cache with 64 total data words with 1 word/cache line, which uses a LRU replacement strategy and a write-back write strategy. This cache architecture is used for parts (C) through (F).

(C) If cache line number 5 is valid and its tag field has the value 0x1234, what is the address in main memory of the data word currently residing in cache line 5?



Main memory address of data word in cache line 5: 0x 0x123414

The program shown on the right repeatedly executes an inner loop that sums the 16 elements of an array that is stored starting in location 0x310.

The program is executed for many iterations, then a measurement of the cache statistics is made during one iteration through all the code, i.e., starting with the execution of the instruction labeled `outer_loop:` until just before the next time that instruction is executed.

```
  . = 0
outer_loop:
  CMOVE(16,R0)    // initialize loop index J
  CMOVE(0,R1)

loop:             // add up elements in array
  SUBC(R0,1,R0)   // decrement index
  MULC(R0,4,R2)   // convert to byte offset
  LD(R2,0x310,R3)// load value from A[J]
  ADD(R3,R1,R1)   // add to sum
  BNE(R0,loop)    // loop until all words are summed

  BR(outer_loop) // perform test again!
```

(D) In total, how many instruction fetches occur during one complete iteration of the outer loop? How many data reads?

ifetch = 2 + (5*16) + 1

dread = 1*16

Number of instruction fetches: 83

Number of data reads: 16

(E) How many instruction fetch misses occur during one complete iteration of the outer loop? How many data read misses? Hint: remember that the array starts at address 0x310.

4 locations in array conflict w/ 4 instructions
⇒ other insts. in cache
⇒ other data in cache

Number of instruction fetch misses: 4

Number of data read misses: 4

(F) What is the hit ratio measured after one complete iteration of the outer loop?

total accesses = 83 + 16 = 99

total hits = 99 - 8

Hit ratio: $\frac{99-8}{99} = \frac{91}{99}$

Problem 2.

The Beta Engineering Team is working on the design of a cache. They've decided that the cache will have **a total of 2^{10} = 1024 data words**, but are still thinking about the other aspects of the cache architecture.

First assume the team chooses to build a direct-mapped write-back cache with a block size of 4 words. ⇒ 4 bit offset

(A) Please answer the following questions:

$\frac{1024 \text{ words}}{4 \text{ wds/line}} = 256$ lines

Number of lines in the cache: 256

Number of bits in the tag field for each cache entry: 20

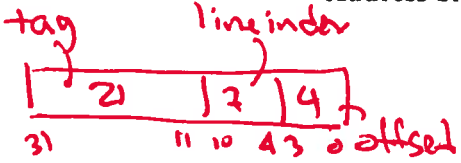
(B) This cache takes *2 clock cycles* to determine if a memory access is a hit or a miss and, if it's a hit, return data to the Beta. If the access is a miss, the cache takes *20 additional clock cycles* to fill the cache line and return the requested word to the Beta. If the hit rate is 90%, what is the Beta's average memory access time in clock cycles?

Average memory access time assuming 90% hit rate (clock cycles): 4

AMAT = 2 + (1 - .9)(20) = 2 + 2 = 4

Now assume the team chooses to build a 2-way set-associative write-back cache with a block size of 4 words. *The total number of data words in the entire cache is still 1024.* The cache uses a LRU replacement strategy. ⇒ 128 lines in each way (7 bits of index)

(C) Please answer the following questions:



Address bits used as offset (including byte offset): A[3 : 0]

Address bits used as cache line index: A[10 : 4]

Address bits used for tag comparison: A[31 : 11]

(D) To implement the LRU replacement strategy this cache requires some additional state for each set. How many state bits are required for each set?

Number of state bits needed for each set for LRU: 1

To test this set-associative cache, the team runs the benchmark code shown on the right. The code sums the elements of a 16-element array. The first instruction of the code is at location 0x0 and the first element of the array is at location 0x10000. Assume that the cache is empty when execution starts and remember *the cache has a block size of 4 words.*

```
. = 0x0
  CMOVE(0,R0)
  CMOVE(0,R1)
L: LD(R0,A,R2)
  ADD(R2,R1,R1)
  ADDC(R0,4,R0)
  CMPLTC(R0,64,R2)
  BT(R2,L)
  HALT()

. = 0x10000
A: LONG(1)
  LONG(2)
  ...
  LONG(15)
  LONG(16)
```

(E) How many instruction misses will occur when running the benchmark?

↳ 2 cache lines

Number of instruction misses when running the benchmark: 2

(F) How many data misses (i.e., misses caused by the memory access from the LD instruction) will occur when running the benchmark?

Number of data misses when running the benchmark: 4

16-element array ⇒ 4 cache lines

(g) What's the exact hit rate when the complete benchmark is executed?

Benchmark hit rate: $\frac{99-6}{99} = \frac{93}{99} \simeq 94\%$

• # inst. fetches = 2 + 16*5 + 1 = 83 (CMOVEs, HALT, ↑ loop iterations)

• # data fetches = 16 ⇒ total fetches = 99

Problem 3.

The program from the Cache performance lab is shown at the right. Assume the program is being run on a Beta with a cache with the following parameters:

- **2-way set-associative**
- **block size of 2**, i.e., 2 data words are stored in each cache line
- total number of data words in the cache is **32**
- **LRU** replacement strategy

```
I = 0x240          // location of program
A = 0x420          // location of array A
N = 16             // size of array (in words)

. = I              // start program here
test:
240 CMOVE(N,R0)    // initialize loop index J
244 CMOVE(0,R1)

loop:              // add up elements in array
248 SUBC(R0,1,R0)  // decrement index
24c MULC(R0,4,R2)  // convert to byte offset
    LD(R2,A,R3)    // load value from A[J]
    ADD(R3,R1,R1)  // add to sum
    BNE(R0,loop)   // loop N times

    BR(test)       // perform test again!

// allocate space to hold array
. = A
    STORAGE(N)     // N words
```

(A) The cache will divide the 32-bit address supplied by the Beta into three fields: B bits of block offset (including byte offset bits), L bits of cache line index, and T bits of tag field. Based on the cache parameters given above, what are the appropriate values for B, L, and T?

8 bytes/line **value for B:** 3

8 lines/way **value for L:** 3

32-L-B **value for T:** 26

(B) If the MULC instruction is resident in a cache line, what will be its cache line index? the value of the tag field for the cache?

0x24C = 0010 0100 1100
offset: 0b 100
line: 0b001
tag: 0b1001

Cache line index for MULC when resident in cache: 1

Tag field for MULC when resident in cache: 0x 9

(C) With the values of I, A, and N as shown, list *all* the values j ($0 \le j < N$) where the location holding the value A[j] will map to the same cache line index as the MULC instruction in the program.

List all j where A[j] have the same cache line index as MULC: 10, 11

A[0] @ 0x420 ⇒ cache line 4. 2 words/line ⇒ A[8], A[9] are in cache line 4.

(D) If the outer loop is run many times, give the steady-state hit ratio for the cache, i.e., assume that the number of compulsory misses as the cache is first filled are insignificant compared to the number of hits and misses during execution.

everything fits in the cache!

Steady-state hit ratio (%): 100%

Problem 4.

Consider a 2-way set-associative cache where each way has 4 cache lines with a **block size of 2 words**. Each cache line includes a valid bit (V) and a dirty bit (D), which is used to implement a write-back strategy. The replacement policy is least-recently-used (LRU). The cache is used for both instruction fetch and data (LD,ST) accesses. Please use this cache when answering questions (A) through (D).

(A) Using this cache, a particular benchmark program experiences an average memory access time (AMAT) of 1.3 cycles. The access time on a cache hit is 1 cycle; the miss penalty (i.e., additional access time) is 10 cycles. What is the hit ratio when running the benchmark program? You can express your answer as a formula if you wish:

Hit ratio for benchmark program: 0.97

$1.3 = 1 + (1 - \text{hit ratio})(10)$

(B) The circuitry for this cache uses various address bits as the block offset, cache line index and tag field. Please indicate which address bits A[31:0] are used for each purpose by placing a "B" in each address bit used for the block offset, "L" in each address bit used for the cache line index, and "T" in each address bit used for the tag field.

Fill in each box with "B", "L", or "T"

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T																										T	L	L	B	0	0

... all T's ...

(C) This cache needs room to store new data and based on the LRU replacement policy has chosen the cache line whose information is shown to the right for replacement. Since the current contents of that line are marked as dirty (D = 1), the cache must write some information back to main memory. What is the address of each memory location to be written? Please give each address in hex.

Way: 0
Cache line index: 3
Valid bit (V): 1
Dirty bit (D): 1
Tag field: 0x123

Addresses of each location to be written (in hex): 0x2478, 0x247C

~~0001 0010 0011~~

using fields from (B)

B: 100, 000

INDEX = 0b11

TAG: 0b 0001 0010 0011

ADDRESS = 0b 10 0100 0111 1X00

(D) This cache is used to run the following benchmark program. The code starts at memory address 0; the array referenced by the code has its first element at memory address 0x2000. First determine the number of memory accesses (both instruction and data) made during each iteration through the loop. Then estimate the steady-state average hit ratio for the program, i.e., the average hit ratio after many iterations through the loop.

```
. = 0
    CMOVE(0,R0)          // byte index into array
    CMOVE(0,R1)          // initialize checksum accumulator
loop:
    LD(R0,array,R2)      // load next element of array
    SHLC(R1,1,R1)        // shift checksum
    ADDC(1,R1,R1)        // increment checksum
    ADD(R2,R1,R1)        // include data value in checksum
    ADDC(R0,4,R0)        // byte index of next array element
    CMPLTC(R0,1000,R2)   // process 250 entries
    BT(R2,loop)
    HALT()

. = 0x2000
array:
    … array contents here …
```

each iteration: 7 ifetches, 1 data access
- instructions occupy 4 cache lines (2 words/line)
  - ↳ 2-way ⇒ 100% hit on ifetch
- data accesses: new word every cycle
  - ↳ 2 words/line ⇒ 50% hit

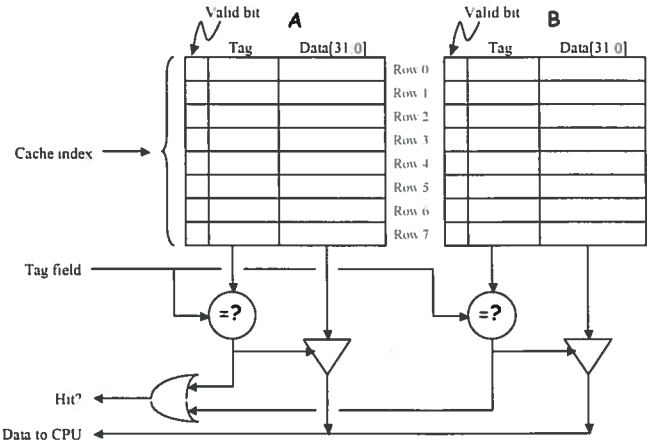
Number of memory accesses made during each iteration of the loop: 8

Estimated steady-state average hit ratio: 15/16 = 93.75%

1 miss every two iterations

Problem 5.

Consider the diagram to the right for a 2-way set associative cache to be used with our Beta design. Each cache line holds a single 32-bit word of data along with its associated tag and valid bit (0 when the cache line is invalid, 1 when the cache line is valid).



(A) The Beta produces 32-bit byte addresses, A[31:0]. To ensure the best cache performance, which address bits should be used for the cache index? For the tag field?

8 cache lines ↳

address bits used for cache index: A[4 : 2]

address bits used for tag field: A[31 : 5]

↑ remaining bits are tag.

(B) Suppose the Beta does a read of location 0x5678. Identify which cache location(s) would be checked to see if that location is in the cache. For each location specify the cache section (A or B) and row number (0 through 7). E.g., **3A** for row 3, section A. If there is a cache hit on this access what would be the contents of the tag data for the cache line that holds the data for this location?

0x5678
0b0101100111100 **cache location(s) checked on access to 0x5678:** 6A, 6B
2 B 3 (offset
tag line index = 0x6 **cache tag data on hit for location 0x5678 (hex): 0x** 2B3

(C) Assume that checking the cache on each read takes 1 cycle and that refilling the cache on a miss takes an *additional* 8 cycles. If we wanted the *average* access time over many reads to be 1.1 cycles, what is the minimum hit ratio the cache must achieve during that period of time? You needn't simplify your answer.

$1.1 = 1 + (1 - HR) \cdot (8)$

minimum hit ratio for 1.1 cycle average access time: 7.9/8

(D) Estimate the approximate cache hit ratio for the following program. Assume the cache is empty before execution begins (all the valid bits are 0) and that an LRU replacement strategy is used. Remember the cache is used for both instruction and data (LD) accesses.

LINE	ADDR	
0	0	`. = 0`
1	4	`CMOVE(source,R0)`
2	8	`CMOVE(0,R1)`
3	C	`CMOVE(0x1000,R2)`
4	10	`loop: LD(R0,0,R3)`
5	14	`ADDC(R0,4,R0)`
6	18	`ADD(R3,R1,R1)`
7	1C	`SUBC(R2,1,R2)`
0	20	`BNE(R2,loop)`
1	24	`ST(R1,source)`
		`HALT()`
		`. = 0x100`
		`source:`
		`. = . + 0x4000 // Set source to 0x100, reserve 1000 words`

each iteration
- 5 inst. fetches ≅ 100% hit
- 1 data fetch ≅ 0% hit
  - no data word fetched twice

6 total

approximate hit ratio: 5/6

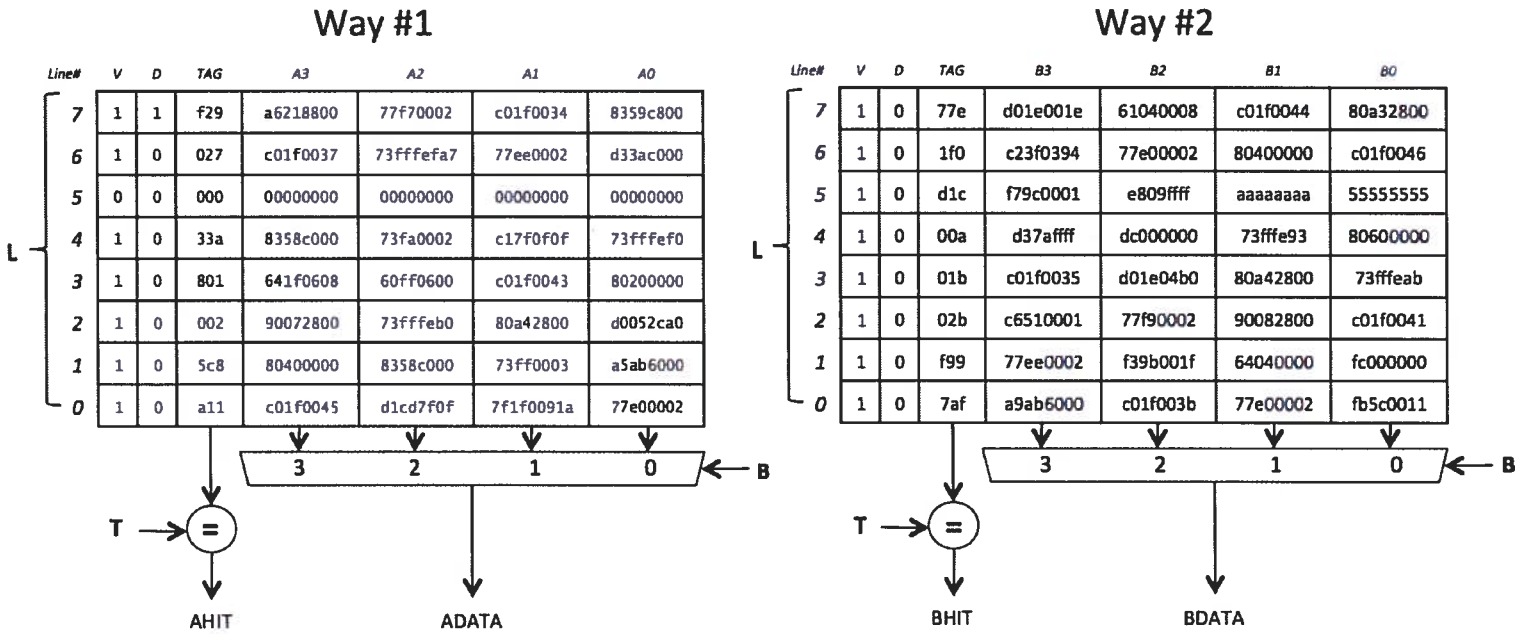
(E) After the program of part (D) has finished execution what information is stored in row 4 of the cache? Give the addresses for the two locations that are cached (one in each of the sections) or briefly explain why that information can't be determined.

Addresses whose data is cached in "Row 4": 0x10 **and** 0x40F0 (data)

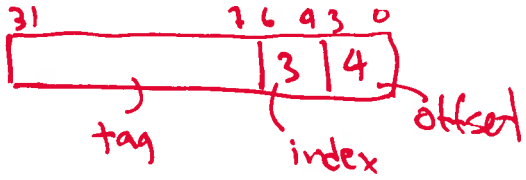
ADDR	40E0	40E4	40E8	40EC	40F0	40F4	40F8	40FC
LINE	0	1	2	3	4			

Problem 6.

A standard unpipelined Beta is connected to a 2-way set-associative cache containing 8 sets, with a block size of 4 32-bit words. The cache uses a LRU replacement strategy. At a particular point during execution, a snapshot is taken of the cache contents, which are shown below. All values are in hex; **assume that any hex digits not shown are 0.**



(A) The cache uses bits from the 32-bit byte address produced by the Beta to select the appropriate set (L), as input to the tag comparisons (T) and to select the appropriate word from the data block (B). For correct and optimal performance what are the appropriate portions of the address to use for L, T and B? Express your answer in the form "A[N:M]" for N and M in the range 0 to 31, or write "CAN'T TELL".



Address bits to use for L: A[6:4]

Address bits to use for T: A[31:7]

Address bits to use for B: A[3:2]

(B) For the following addresses, if the contents of the specified location appear in the cache, give the location's 32-bit contents in hex (determined by using the appropriate value from the cache). If the contents of the specified location are NOT in the cache, write "MISS".

B=0, L=0, T=0x1422 **Contents of location 0xA1100 (in hex) or "MISS": 0x** miss

B=2, L=4, T=0xA **Contents of location 0x548 (in hex) or "MISS": 0x** DC000000

(C) Ignoring the current contents of the cache, is it possible for the contents of locations 0x0 and 0x1000 to both be present in the cache simultaneously?

Locations 0x0 and 0x1000 present simultaneously (circle one): (YES) ... **NO**

both map to cache line 0, but it's a 2-way cache

(D) (Give a one-sentence explanation of how the D bit got set to 1 for Line #7 of Way #1.

One sentence explanation

ST changed a value of a word on that cache line but it hasn't yet been written back to memory.

(E) The following code snippet sums the elements of the 32-element integer array X. Assume this code is executing on a Beta with a cache architecture as described above and that, initially, the cache is empty, i.e., all the V bits have been set to 0. Compute the hit ratio as this program runs until it executes the HALT() instruction, a total of 2 + (6*32) + 1 = 195 instruction fetches and 32 data accesses.

Hit ratio: 216/227 = 95.29%

```
        . = 0
+2  {   CMOVE(0, R0)          // loop counter
    {   CMOVE(0, R1)          // accumulated sum

loop:
        SHLC(R0, 2, R2)       // convert loop counter to byte offset
        LD(R2, X, R3)         // load next value from array
        ADD(R3, R1, R1)       // add value to sum
        ADDC(R0, 1, R0)       // increment loop counter
        CMPLTC(R0, 32, R2)    // finished with all 32 elements?
        BT(R2,loop)           // nope, keep going

+1      HALT()                // all done, sum in R1

X:      LONG(1)               // the 32-element integer array X
        LONG(2)
        ...
        LONG(32)
```

6 ifetch + 1 dfetch × 32 iterations

11 misses total out of 227 accesses

- 2-way cache: 100% hit rate on ifetch
- block size = 4:
  - 3 misses to load instructions
  - 32 data accesses, but only 25% miss (each miss loads 4 words) ⇒ 8 data misses total

Problem 7.

After his geek hit single *I Hit the Line,* renegade singer Johnny Cache has decided he'd better actually learn how a cache works. He bought three Beta processors, identical except for their cache architectures:

- ***Beta1*** has a 64-line direct-mapped cache
- ***Beta2*** has a 2-way set associative cache, LRU, with a total of 64 lines
- ***Beta3*** has a 4-way set associative cache, LRU, with a total of 64 lines

Note that each cache has the same total capacity: 64 lines, each holding a single 32-bit **word** of data or instruction. All three machines use the same cache for data and instructions fetched from main memory.

Johnny has written a simple test program:

```
// Try a little cache benchmark
I = 0x1000                      // where program lives
A = 0x2000                      // data region 1
B = 0x3000                      // data region 2
N = 16                          // size of data regions (BYTES!)

. = I                           // start program here
P:    CMOVE(1000, R6)           // outer loop count
Q:    CMOVE(N, R0)              // Loop index I (array offset)
R:    SUBC(R0, 4, R0)           // I = I-1
      LD(R0, A, R1)             // read A[I]
      LD(R0, B, R2)             // read B[I]
      BNE(R0, R)
      SUBC(R6,1, R6)            // repeat many times
      BNE(R6, Q)
      HALT()
```

Johnny runs his program on each Beta, and finds that one Beta model outperforms the other two.

(A) (2 points) Which Beta gets the highest hit ratio on the above benchmark?

3 regions: 0x1000+, 0x2000+, 0x3000+ **Circle one:** **Beta1** **Beta2** **Beta3**

(B) (2 points) Johnny changes the value of **B** in his program to **0x2000** (same as **A**), and finds a substantial improvement in the hit rate attained by one of the Beta models (approaching 100%). Which model shows this marked improvement?

now just 2 regions: 0x1000+, 0x2000+ **Circle one:** **Beta1** **Beta2** **Beta3**

(C) (3 points) Finally, Johnny sets **I**, **A**, and **B** each to **0x0**, and sets **N** to **64**. What is the TOTAL number of cache misses that will occur executing this version of the program on each of the Beta models?

TOTAL cache misses running on Beta1: 16 ; **Beta2:** 16 ; **Beta3:** 16

only compulsory misses for all these caches

6.004 Computation Structures
Spring 2017