6.004 Fall 2020 Recitation Problems
L16 – Caches

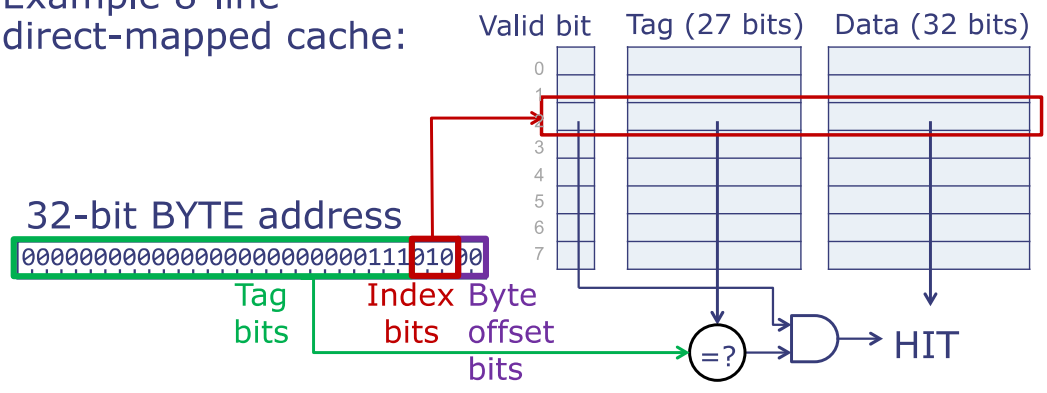
Keep the most often-used data in a small, fast SRAM (often local to CPU chip). The reason this strategy works: LOCALITY.

- *Temporal locality: If a location has been accessed recently, it is likely to be accessed (reused) soon*
- *Spatial locality: If a location has been accessed recently, it is likely that nearby locations will be accessed soon*

AMAT(Average Memory Access Time) = HitTime + MissRatio * MissPenalty

Direct-Mapped Caches

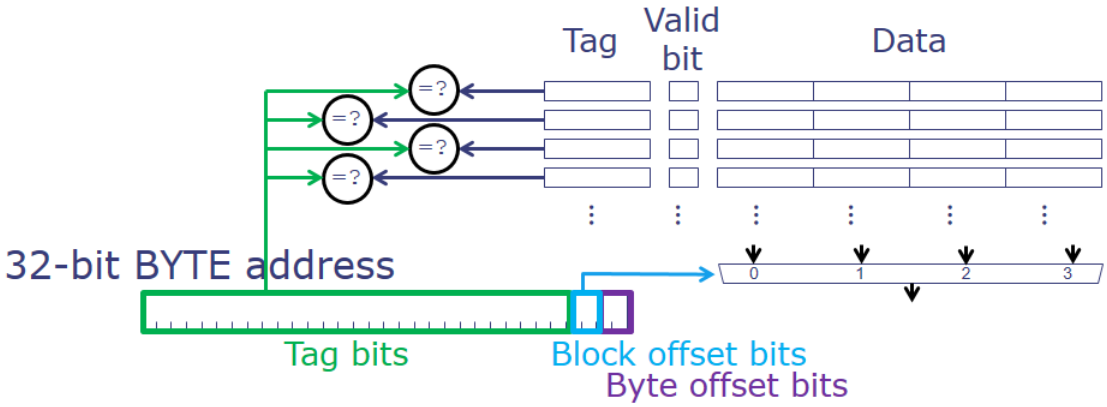
- Each word in memory maps into a single cache line
- Access (for cache with 2^W lines):
  - Index into cache with W address bits (the index bits)
  - Read out valid bit, tag, and data
  - If valid bit == 1 and tag matches upper address bits, HIT
- Example 8-line direct-mapped cache:



Fully-Associative Cache

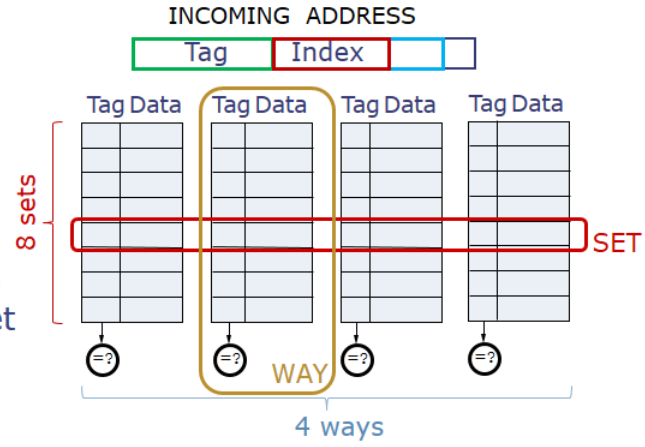
Opposite extreme: Any address can be in any location

- No cache index!
- Flexible (no conflict misses)
- Expensive: Must compare tags of all entries in parallel to find matching one



N-way Set-Associative Cache

- Use multiple direct-mapped caches in parallel to reduce conflict misses
  - Nomenclature:
    - # Rows = # Sets
    - # Columns = # Ways
    - Set size = #ways = "set associativity" (e.g., 4-way → 4 lines/set)
  - Each address maps to only one set, but can be in any way within the set
  - Tags from all ways are checked in parallel



- Fully-associative cache: Extreme case with a single set and as many ways as cache lines

Problem 1 (parts A and B are from worksheet 15).

The RISC-V 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 cache with a block size of 4 words.

(A) Please answer the following questions:

Number of lines in the cache: _____

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

(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 processor. 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 processor. If the hit rate is 90%, what is the processor's average memory access time in clock cycles?

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

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.

(C) Please answer the following questions

Address bits used as block offset: A[_:_]

Address bits used as cache line index: A[_:_]

Address bits used for tag comparison: A[_:_]

(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: _____

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*.

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

Number of instruction misses when running the benchmark: _____

(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: _____

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

Benchmark hit rate: _____

```
. = 0x0
   mv x3, x0 // index
   mv x1, x0 // sum
   // x4 = 0x10000
   lui x4, 0x10

L: add x5, x4, x3
   lw x2, 0(x5)
   add x1, x1, x2
   addi x3, x3, 4
   slti x2, x3, 64
   bnez x2, L
   unimp    // halt

. = 0x10000
A: .word 0x1
   .word 0x2
   …
   .word 0xF
   .word 0x10
```

Problem 2.

Assume, the program shown on the right is being run on a RISC-V processor 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

```
. = 0x240         // start of program
test:
  addi x4, x0, 16 // initialize loop index J
                  // to size of array
  mv x1, x0       // x1: sum

loop:             // add up elements in array
  subi x4, x4, 1  // decrement index
  slli x2, x4, 2  // convert to byte offset
  lw x3, 0x420(x2)// load value from A[J]
  add x1, x1, x3  // add to sum
  bnez x4, loop   // loop N times

  j test          // perform test again!

// allocate space to hold array
. = 0x420
A: .word A[0]
   .word A[1]
   …
```

(A) The cache will divide the 32-bit address supplied by the processor into four fields: 2 bits of byte offset, B bits of block offset, 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?

value for B: _____
value for L: _____
value for T: _____

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

Cache line index for SLLI when resident in cache: _____

Tag field for SLLI when resident in cache: _____

(C) Given that the code begins at address 0x240 and the array begins at address 0x420, and that there are 16 elements in the array as shown in the code above, list *all* the values j ($0 \le j < 16$) where the location holding the value A[j] will map to the same cache line index as the `SLLI` instruction in the program.

List all j where A[j] have the same cache line index as SLLI: _____

(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.

Steady-state hit ratio (%): _____

Problem 3. ★

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:

$1.3 = 1 + \text{missRatio} \times 10$ **Hit ratio for benchmark program:** 0.97

(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.

block size = 2 ⇒ offset bits: 3 (2 for byte offset, 1 for block offset)
4 lines/each way = 4 sets index bits: 2

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																							L	L	B	0	0

address {27'b tag, 26'b index, 1'b block offset, 2'b byte offset}

(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

Tag 0x123
= 0b 0... 0001_0010_0011

Block offset: either ① 0b1 or ② 0b0

① 0b 00...00_0010_0100_0111_1000 = 0x2478
② 0b 00...00_0010_0100_0111_1100 = 0x247C

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

(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 0x200. 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
    mv x3, x0           // byte index into array
    mv x1, x0           // initialize checksum accumulator
loop:
① → lw x2, 0x200(x3)    // load next element of array
```

1 memory access

7 instr

```
    slli x1, x1, 1        // shift checksum
    addi x1, x1, 1        // increment checksum
    add x1, x1, x2        // include data value in checksum
    addi x3, x3, 4        // byte index of next array element
    slti x2, x3, 1000     // process 250 entries
    bnez x2, loop
    unimp                 // halt

. = 0x200
array:
    … array contents here …
```

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

Estimated steady-state average hit ratio: 15/16

7 instr + 1 memory = 8

instr miss: 0 (no conflicts!)

block size: 2

every other loop, miss once for data read

(lw x2, 0x200(x3))

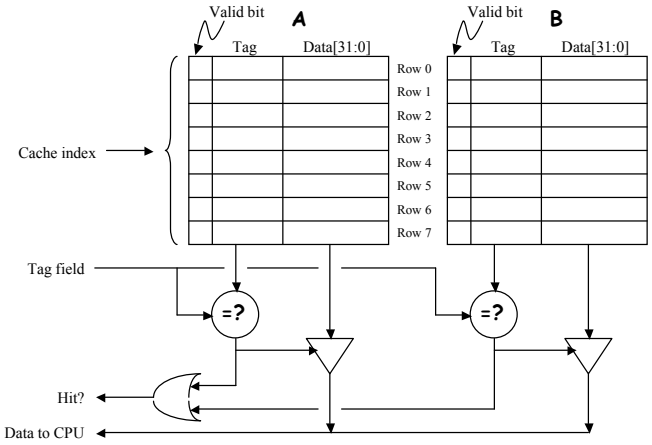
miss: 1/(2×8) = 1/16

↑ accesses/loop

hit ratio: 15/16

Problem 4.

Consider the diagram to the right for a 2-way set associative cache to be used with our RISC-V processor. 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 RISC-V 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?

address bits used for cache index: A[_:_]

address bits used for tag field: A[_:_]

(B) Suppose the processor 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?

cache location(s) checked on access to 0x5678: _____

cache tag data on hit for location 0x5678 (hex): _____

(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.

minimum hit ratio for 1.1 cycle average access time: _____

(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.

```
      . = 0
      addi x4, x0, 0x100
      mv x1, x0
      lui x2, 1            // x2 = 0x1000
loop: lw x3, 0(x4)
      addi x4, x4, 4
      add x1, x1, x3
      addi x2, x2, -1
      bnez x2, loop
      sw x1, 0x100(x0)
      unimp                // halt
```

```
        . = 0x100
source:
        . = . + 0x4000 // Set source to 0x100, reserve 0x1000 words
```

approximate hit ratio: _____

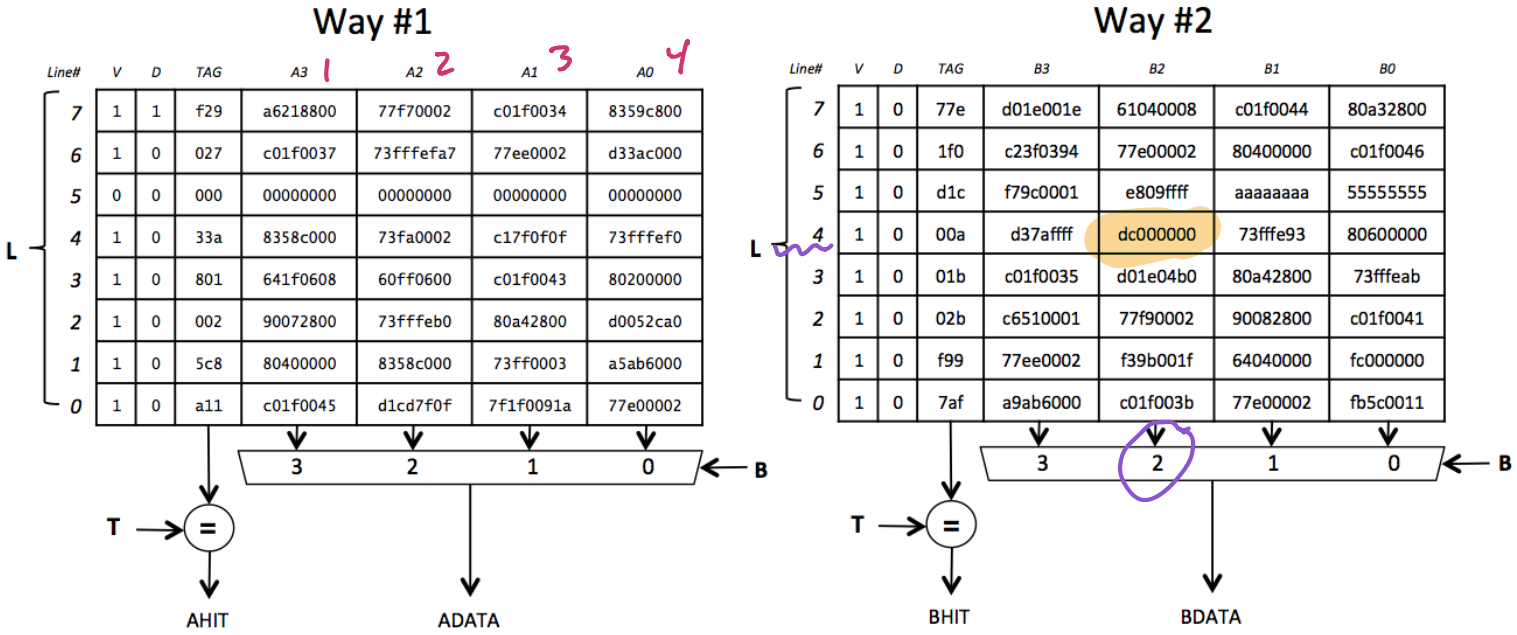
(E) After the program of part (D) has finished execution what information is stored in row 4 of the cache? Give the addresses for the two locations that are cached (one in each of the sections) or briefly explain why that information can't be determined.

Addresses whose data is cached in "Row 4": _____ and _____

a) address = { 25'b tag, 3'b index, 2'b block offset, 2'b byte offset }
[31 : 7] [6 : 4] [3 : 2] [1 : 0]

Problem 5. ★

A standard unpipelined RISC-V 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 processor 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".

4 words/block ⇒ 2 bits for words, 2 bits to bytes } offset bits: 4

8 rows/way → index bit: 3

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] offsets

(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".

0xA1100 = 0b 101 0000 1 000 10 00 00
tag: 0x1 (4) (2) (2) line: 0 wd: 0

0x548 = 0b 101 0100 1000
tag: 0xA word: 2
line: 4

Contents of location 0xA1100 (in hex) or "MISS": MISS

Contents of location 0x548 (in hex) or "MISS": dc 000000

(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

yes because it's 2-ways

(D) Give a one-sentence explanation of how the D bit got set to 1 for Line #7 of Way #1. At what point should the D bit be reset to 0?

One sentence explanation

value of word was updated but hasn't been written to memory.

write back

Will be reset when line evicted from cache & written to memory

(E) The following code snippet sums the elements of the 32-element integer array X. Assume this code is executing on a RISC-V processor 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 *unimp* instruction, a total of 2 + (6*32) + 1 = 195 instruction fetches and 32 data accesses.

195+32=227

Hit ratio: 216/227

```
①  . = 0
   mv x4, x0          // loop counter
   mv x1, x0          // accumulated sum

loop:
   slli x2, x4, 2     // convert loop counter to byte offset
   lw x3, 0x100(x2)   // load next value from array
②  add x1, x1, x3     // add value to sum
   addi x4, x4, 1     // increment loop counter
   slti x2, x4, 32    // finished with all 32 elements?
   bnez x2, loop      // nope, keep going

③  unimp              // all done, sum in x1

. = 0x100
X:  .word 1            // the 32-element integer array X
    .word 2
    …
    .word 32
```

cache stores 4 words/line

- first 4 map to cache index 0
- other 4 map to cache index 1

- miss 8 data
- miss 3 instructions

195 inst

32 data accesses

blocks

32/4 = 8

block size

⇒ miss 1 every 4

227 − 8 − 3 = 216

Problem 6. ★

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 RISC-V processors, identical except for their cache architectures:

- ***Proc1*** has a 64-line direct-mapped cache
- ***Proc2*** has a 2-way set associative cache, LRU, with a total of 64 lines
- ***Proc3*** 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 progr

```
// Try a little cache benchmark
// Assume x7 = 0x2000 (data region A)
// Assume x8 = 0x3000 (data region B)
// Assume x9 = 16 (size of data regions in BYTES!)

. = 0x1000                      // start program here
P:    addi x6, x0, 1000         // outer loop count
Q:    mv x3, x9                 // Loop index i (array offset)
R:    addi x3, x3, -4           // i = i-1
      addi x9, x3, x7           // x9 = address of A[i]
      addi x10, x3, x8          // x10 = address of B[i]
      lw x1, 0(x9)              // read A[i]
      lw x2, 0(x10)             // read B[i]
      bnez x3, R
      addi x6, x6, -1           // repeat many times
      bnez x6, Q
      unimp                     // halt
```

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

(A) Which processor model gets the highest hit ratio on the above benchmark?

Circle one: **Proc1** **Proc2** **Proc3**

(B) 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 processor models (approaching 100%). Which model shows this marked improvement?

Circle one: **Proc1** **Proc2** **Proc3**

(C) Finally, Johnny moves the code region to 0x0 and the two data regions **A**, and **B** each to **0x0000** and sets **x9** to **64**. What is the TOTAL number of cache misses that will occur executing this version of the program on each of the processor models?

TOTAL cache misses running on Proc1: 64 ; **Proc2:** 64 ; **Proc3:** 64

Problem 7. ★

(A) We would like to design a cache with an AMAT (average memory access time) of 1.5 cycles. Accessing our cache takes 1 cycle, and on a miss, it takes an *additional* 10 cycles to retrieve the data from main memory and update the cache. What does our hit ratio need to be in order to achieve the target AMAT?

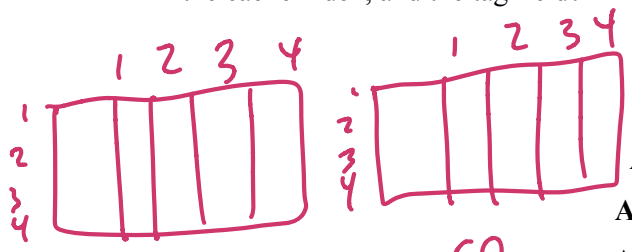
$1.5 = 1 + \text{miss ratio} \times 10$

$\Rightarrow$ hit ratio $= 95\%$

Hit ratio = 0.95

We choose to implement a 2-way set-associative cache with a block size of 4 (i.e. 4 words per line). The number of sets in the cache is 4. Assume that addresses and data words are 32 bits.

(B) To ensure the best cache performance, which address bits should be used for the block offset, the cache index, and the tag field?



$4 = 2^{(2)}$ index bits

Address bits used for byte offset: A[1 : 0]

Address bits used for block offset: A[3 **:** 2 **]**

Address bits used for cache index: A[5 **:** 4 **]**

Address bits used for tag field: A[31 **:** 6 **]**

(C) Assuming the cache uses a writeback policy, what is the total number of bits per cache line? Please show your work for partial credit.

1 valid bit (whether this is storing any data) + 1 dirty bit (whether consistent w/ main memory) + 26 tag + 4 × 32 data bits

156 **bits**

$1 + 1 + 26 + 4 \times 32 = 156$

We want to analyze the performance of this cache on the following assembly program, which iterates through a 1000-word array `A` and sets each element to `A[i] = -A[i]`. The base address of array A is `0x3000`.

```
. = 0x100 // The following code starts at address 0x100

// Assume the following registers are initialized:
// x1=0 (loop index)
// x2=1000 (number of array elements)
// x3=0x3000 (base address of array A)
loop:
    slli x5, x1, 2  // x5 = byte offset of the ith element
    add x6, x5, x3  // x6 = address of A[i]
    lw x7, 0(x6)    // x7 = A[i]
    sub x7, x0, x7  // x7 = -A[i]
    sw x7, 0(x6)    // store A[i] = -A[i]
    addi x1, x1, 1  // increment i
    blt x1, x2, loop// continue looping
```

(D) Below is the cache state the first time the program is about to enter the loop at `loop`. Assume that the cache uses a least-recently used (LRU) replacement policy, and that all cache lines in Way 1 are currently the least-recently used. Mark up the cache below to indicate the state of the cache immediately after one loop iteration (after executing the `blt` instruction for the first time). You do not need to specify the value of data words, but do specify the values of D (dirty bit), V (valid bit), and Tag.

Way 0

D	V	Tag	Word 0	Word 1	Word 2	Word 3
0	1	0x0				
0	1	0x0				
0	1	0x0				
-	0	-				

Way 1

D	V	Tag	Word 0	Word 1	Word 2	Word 3
-	0	-				
-	0	-				
-	0	-				
-	0	-				

(E) How many instruction fetches and data accesses occur per iteration of the loop?

Number of instruction fetches: 7

Number of data accesses: 2

(F) After the program has been running for many loop iterations, what is the steady-state hit ratio for instruction fetches and data accesses?

1 miss 0 misses

Steady-state hit ratio for instruction fetches: 1 (100%)

Steady-state hit ratio for data accesses: 7/8

1 miss every 4 loop

lw & sw every loop

⇒ 1/8 miss

Problem 8.

Anne and Ben just learned about caches in 6.004 and decided to design their own.

(A) They would like to design a cache with an AMAT (average memory access time) of 3 cycles. Accessing the cache should take 1 cycle, and on a miss, it should take an *additional* 16 cycles to retrieve the data from main memory, update the cache, and return the requested word to the processor. What should their hit ratio be in order to achieve the target AMAT?

Hit ratio: _____

Ben suggests implementing a 2-way set-associative cache with a block size of 4 (i.e. 4 words per line). The number of sets in the cache is 8. Assume that addresses and data words are 32 bits wide.

(B) To ensure the best cache performance, which address bits should be used for the block offset, the cache index, and the tag field?

Address bits used for byte offset: A[__1__ : __0__]

Address bits used for block offset: A[_____ : _____]

Address bits used for cache index: A[_____ : _____]

Address bits used for tag field: A[_____ : _____]

(C) Anne agrees with the appropriateness of a 2-way set-associative implementation, but she suspects that a larger block size might result in a higher hit rate. Suppose the block size of the cache is doubled to 8. If the total number of data words in the cache remains unchanged, how would the number of cache lines change?

Change in # of cache lines (select one of the choices below):

UNCHANGED … +1 … -1 … 2x … 0.5x … CAN'T TELL

Ultimately, they choose to implement Ben's 2-way, 4-block cache. Below is a snapshot of the cache during the execution of some unknown code. The column labeled *Word x* corresponds to the x^{th} word of the block. The V bit specifies whether or not the line is valid, and the D bit specifies whether or not the line is dirty.

	V	D	Tag	Word 0	Word 1	Word 2	Word 3
set 0	1	0	0x32	0x0A	0x1A	0x2A	0x3A
set 1	1	0	0x32	0x4B	0x5B	0x6B	0x7B
set 2	1	0	0x32	0x3C	0x2C	0x1C	0x0C
set 3	1	0	0x32	0x7D	0x6D	0x5D	0x4D
set 4	1	1	0x50	0x33	0x23	0x13	0x03
set 5	1	1	0x50	0x44	0x34	0x24	0x14
set 6	0	0	0x43	0x55	0x65	0x75	0x85
set 7	1	0	0x66	0x66	0x76	0x86	0x96

V	D	Tag	Word 0	Word 1	Word 2	Word 3
1	0	0x33	0x80	0x81	0x82	0x83
1	1	0x33	0xB4	0xB5	0xB6	0xB7
1	0	0x95	0xC3	0xC2	0xC1	0xC0
1	0	0x95	0xD3	0xD4	0xD5	0xD6
1	0	0x22	0x89	0x88	0x87	0x86
1	0	0xA0	0x92	0x93	0x94	0x95
1	1	0x37	0xF5	0xF6	0xF7	0xF8
1	1	0x18	0xA7	0xA8	0xA9	0xAA

(D) Would a load request to address 0x193C result in a hit or a miss? If it results in a hit, specify what value is returned; if it is a miss, write N/A.

Hit / Miss : _____

Returned value if hit or N/A if miss: _____

Anne and Ben want to analyze the performance of this cache on the following assembly program, which calculates the first 256 terms in the Fibonacci sequence and stores them in an array `A`. The base address of array `A` is `0x3000`.

```
// Assume the following registers are initialized:
// x1 = 0 (initial loop index)
// x2 = 256 - 2 = 254 (number of Fibonacci elements to calculate)
// x3 = 0x3000 (base address of array A)

. = 0x100 // The following code starts at address 0x100
fibonacci:
   li x4, 1         // x4 = 1 (second element in sequence)
   sw x0, 0(x3)     // A[0] = 0
   sw x4, 4(x3)     // A[1] = 1
loop:
   slli x4, x1, 2   // x4 = byte offset of the ith element
   add x5, x4, x3   // x5 = address of A[i]
   lw x6, 0(x5)     // x6 = A[i]
   lw x7, 4(x5)     // x7 = A[i+1]
   add x6, x6, x7   // x6 = A[i] + A[i+1]
   sw x6, 8(x5)     // A[i+2] = x6
   addi x1, x1, 1   // increment i
   blt x1, x2, loop // continue looping
```

Answer the following questions about the behavior of the cache during execution of the above code. Assume that the cache uses a least recently used (LRU) replacement policy, that the cache is initially empty, and that all cache lines in Way 0 are currently the least-recently used.

(E) How many instruction fetches and data accesses occur per iteration of the loop?

Number of instruction fetches: _____

Number of data accesses: _____

(F) After the program has been running for many loop iterations, what is the steady-state hit ratio for instruction fetches and data accesses? *Hint: Note that in steady state each array element is accessed in multiple loop iterations.*

Steady-state hit ratio for instruction fetches: _____

Steady-state hit ratio for data accesses: _____