3. **[17 points] Caches**

(a) Consider an LC4 processor with an L1 data cache that is 16KB, 2-way set-associative and has 4B blocks with an LRU eviction policy. Give the number of bits in each of the *tag*, *index* and *block offset* for this cache.

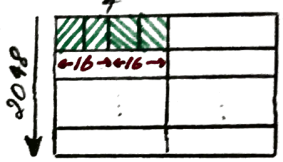
[3 points] 4b tag, 11b index, 1b block offset (LC4 is 16-bit, *not* byte, addressable)

tag bits: index bits: block offset bits:

$\#blocks = \frac{16KB}{4B} = \frac{4096}{2\text{-way}} = 2048$ per way

index bits: $\log_2(2048) = 11$ bits

block offset = $\log_2(2) = 1$ bit



(b) Given a cache (for a *byte-addressable, 16-bit processor*) with a **1B block size**, what capacity and associativity would result in a cache where the number of tag bits equals the number of bits in each block?

[4 points] Need 8 index bits, so capacity is 256B * associativity.

capacity (bytes): associativity:

(c) The sequence below shows a series of accesses to an initially-empty 1KB 2-way set associative cache with an LRU replacement policy. This cache is part of an LC4 processor. Given this sequence, what do we know about the block size of the cache?

[10 points] block size must be 16B (3b block offset * 2B words)

address	result
0101 0001 0100 0000	miss
1000 0000 0011 0010	miss
0101 0001 0100 0010	hit
0110 0000 1100 0100	miss
1001 0001 0000 0011	miss
0101 0001 0100 0100	hit
1000 0000 0011 0001	hit
0101 0001 0100 1000	miss

block size (bytes):

3. **[11 points] Caches**
Consider an LC4 processor with an L1 data cache that is 8KB, 2-way set-associative and has 8B blocks with an LRU eviction policy.

(a) Give the number of bits in each of the *tag*, *index* and *block offset* for this cache.

Tag bits: 5 bits

Index bits: 9 bits

Block offset bits: 3 bits since LC4 is 16 bit addressable

- Elaborate on this calculation

(b) Fill in the missing entries in the table below for a series of accesses to the cache described above.

address	hit or miss?
0010 1000 1000 0010	hit
1000 0010 0000 0010	miss
0010 1000 1000 0100	miss
0110 0010 0000 0001	miss
1001 0010 0000 0110	miss
1000 0010 0000 0010	hit
0011 1000 1000 0111	miss
1001 0010 0000 0101	hit
1001 0010 0000 0101	hit

0.) The Easy One (1 point total)

(a) Check to make certain that your exam has all 5 pages (excluding the cover sheet).

(b) Write your name and PennKey (username) on the front of the exam. Leave the recitation blank.

(c) Sign the certification that you comply with the Penn Academic Integrity Code.

1.) Performance Pressure (9 points total)

For each of the scenarios on the left, list the feature(s) on the right that definitely make sense to incorporate into your processor design. Do not include features whose impact is impossible to determine. For instance, list "small caches" or "big caches" only if there is a clear advantage to one or the other, but not if the cache size is totally irrelevant. The first scenario is completed for you.

(a) **Space Heater:** Your heat isn't working so you're running an infinite loop of random instructions that keeps the processor busy while you huddle close to it for warmth and to toast marshmallows: **b, c, d, j, k, l**

(b) **Hearing Aid:** You need to stream audio data from three microphones, separate speech from background noise, and play the enhanced audio stream through a speaker. Performance must be completely consistent to avoid skips and clicks in the output. The processor and battery must fit comfortably behind the user's ear, placing severe constraints on available power and cooling.

(a) Small cache (i) Predicated instructions
(c) 0 or 1 level cache (e) Write-back cache

(a) Small caches
(b) Big caches
(c) 0 or 1 levels of cache
(d) 2 or more levels of cache
(e) Write-back cache
(f) Write-through cache
(g) Inclusive cache
(h) Exclusive cache
(i) Predicated instructions
(j) Fancy branch predictor
(k) Superscalar pipeline
(l) Out-of-order pipeline

(c) **Ticketmaster:** You're running a web server that sells tickets to concerts and shows. Most of shows have low volume, but a few blockbusters generate a mad rush of customers trying to buy tickets all at once. For simplicity, assume the web server handles all requests from a single thread of execution rather than spawning a new process for each request.

(b) Big cache (e) Write back cache (j) Fancy branch predictor
(d) 2 or more level cache (i) Predicated instruction (k) Superscalar (l) out of order

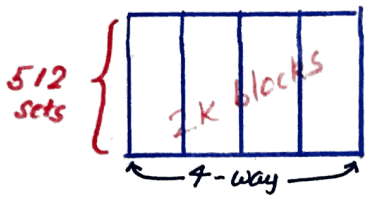
(d) **Web Crawler:** You're computing the frequency count of every word on every web page in existence as part of generating the latest, greatest search engine. (There is now so much drivel on the web that indexing must be spread across hundreds of systems. You're system is only responsible for frequency counts.) All you're really doing is loading HTML files one by one, computing a hash of each word on the page, and incrementing a counter stored in the hash bucket. But you have to do this extra fast in order to keep up with the rest of the indexing process.

(b) 2 or more levels of cache (e) Write back cache (l) out of order
(d) Big cache (k) Superscalar

CIS 501 – Computer Architecture	Final Exam
Prof. Devietti	December 20, 2017

2. [12 points] Caches

(a) In a 32KB 4-way set-associative cache with 16B blocks, how many **sets** are there? How many **blocks**?



$$\text{No. of blocks} = \frac{\text{Total Cache size}}{\text{Block size}} = \frac{32KB}{16B} = 2K$$

$$\text{No. of sets} = \frac{\#\text{Blocks}}{\text{Associativity}} = \frac{2K}{4} = 512$$

(b) What is the defining feature of a cache with zero block offset bits?

Each block / line is only 1B

(c) What is the defining feature of a cache with zero index bits?

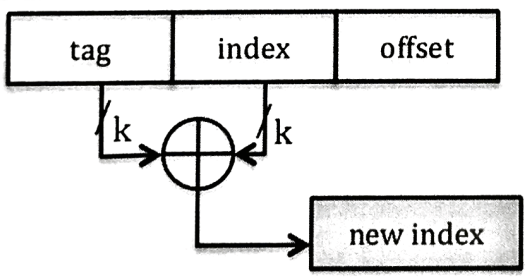
Fully Associative Cache

(d) What is the defining feature of a cache with zero tag bits? Assume 32-bit addresses.

(i) Direct Mapped Cache

(ii) The cache is 2^{32} Bytes = 4GB, since each address is uniquely mapped to the cache.

(e) Dr. Otto Order proposes a new cache indexing mechanism that, given a cache that uses *k* index bits, XORs the *k* lowest-order bits of the conventional tag with the conventional index bits to produce a new index. The cache stores the conventional tag bits as normal. Will this cache work correctly? What behavior, if any, does it have that's different from a conventional cache?



(i) This will work.

(ii) But will have different aliasing / conflicts than a regular cache.

(iii) Parallel TAG and INDEX access is not possible anymore, could be slower

5. [6 points] Caches

(a) What are the two kinds of locality that caches exploit?

i) Temporal Locality

ii) Spatial Locality

(b) Code that accesses every element of an array in sequence exhibits what kind of locality?

Spatial Locality

(c) Consider a 32-bit processor, with a 32KB cache with 16B blocks. How many bits are in the tag, index and offset for this cache?

$$\#\text{Blocks} = \frac{\text{Total cache size}}{\text{Block size}} = \frac{32\text{KB}}{16\text{B}} = 2048$$

Offset Bits: $\log_2(16) = 4$ bits

Index Bits: $\log_2(2048) = 11$ bits

Tag Bits: $32 - 11 - 4 = 17$ bits

5. [18 points] Caches

For the following questions, consider a system with 16-bit addresses, and a 8KB direct-mapped cache with 32B blocks.

(a) How many bits are in the tag, index and block offset for this cache?

[3 points] 3-bit tag, 8-bit index, 5-bit block offset

(b) What is the size of the tag array (in bytes) for this cache? Be sure to account for the valid bits.

[2 points] There are 256 blocks in the cache, and each tag array entry is 4 bits (3 bit tag + valid bit) so the total tag storage is 128 bytes.

Full credit if calculating correctly using answers from (a).

+1 point for calculating total tag entry

+1 point for the total storage calculation

-1 point if missing bits=>bytes conversion

(c) Give a sequence of 2 addresses that, if accessed in a loop repeatedly, will result in a 100% miss rate in the above cache.

[4 points] We want two addresses that map to the same set, i.e., have the same index bits but different tags.

000 00000000 00000

001 00000000 00000

(d) Consider an alternate 16KB direct-mapped cache with 8B blocks. Give a sequence of two addresses that, if accessed in a loop repeatedly, would result in a 100% miss rate in **both** this 16KB cache **and** the 8KB direct-mapped cache with 32B blocks from before.

[4 points] We want two addresses that have the same index bits in both caches, but different tags. The 16KB cache has 3 block offset bits, 11 index bits, and 2 tag bits. So we want the addresses to differ solely in the 2 highest-order bits (which are tag bits for both caches).

00 0000 0000 0000 00

01 0000 0000 0000 00

(e) Dr. Chip D. Signer has proposed a new direct-mapped cache design that avoids many cache conflicts without requiring set associativity. Dr. Signer's proposal is to xor the branch history register with the index bits, and use the result to identify which cache set to use. This way, he argues, conflicts can be avoided just like in a gshare branch predictor.

Is Dr. Signer's proposal a good idea? If so, describe one benefit of the cache design. If not, give a specific example of something bad that can happen with the design.

[5 points] This is not a good idea. Two accesses to the same address A may map to different sets if the BHR has different values at each access. This means that a store to A may not be reflected in a subsequent load of A, breaking program correctness.

3. [12 points] Caching

(a) [3 points] Dr. Meg Ahertz is considering a new writeback cache design that has one dirty bit per byte, instead of the conventional one dirty bit per line. Dirty bits are now set based on the actual bytes written by a store instruction.

i) Give one advantage of Dr. Ahertz's scheme.

The total amount of information that will be written back will reduce, therefore reducing writeback traffic.

ii) Give one disadvantage of Dr. Ahertz's scheme.

More space ⇒ slower cache

(b) [3 points] Dr. Chip D. Signer hears about Dr. Ahertz's design and decides to go one better and have a single dirty bit per cache set in a set-associative cache. Is this a good idea? Why or why not?

(i) No

(ii) You will have to writeback the entire set whenever ANY eviction occurs - significantly increasing writeback traffic.

(c) [6 points] Consider an 8-bit uniprocessor cache hierarchy with a direct-mapped 8B L1 cache, and a direct-mapped 16B inclusive L2 cache. All cache lines are 4B. All caches are initially empty. For each binary address in the trace below, state whether it is an *L1 hit*, an *L2 hit*, or a *miss*.

address	outcome
1100 0110	miss
0100 1111	miss
1100 0101	L2 hit
1001 0100	miss
1100 0110	miss

1. How to pick offset/index/tag?

2. Spatial locality - bring entire block?

2. [14 points] Caches

(a) [1 point] What is the key advantage provided by a store buffer?

Hides store miss / write miss latency

(b) [1 point] What is one advantage provided by a victim buffer?

(i) Reduces conflict misses
(ii) Increases associativity with minimum overhead

(c) [6 points] Name the three C's in the 3C characterization of cache misses. For each kind of miss, describe a hardware change that can help reduce misses of that kind.

i) Compulsory Misses - Bigger blocks and prefetching

ii) Capacity Misses - Increase cache sizes

iii) Conflict Misses - Add associativity

(d) [6 points] Consider two different cache configurations for an 8-bit processor. Both caches have four 16-byte blocks (for a total capacity of 64 bytes), but one is direct-mapped and the other is two-way set associative and uses the LRU replacement algorithm. Both caches are initially empty.

i) Give a short sequence (4 or fewer) of addresses in which **the set-associative cache has a better hit rate** than the direct-mapped cache. (Give the addresses as 8-digit binary numbers.) Also give the hit rate for each cache.

0000 0000
1000 0000
0000 0000

Set Associative: 33% hit rate
Direct Mapped: 0% hit rate

ii) Give a short sequence (4 or fewer) of addresses in which **the direct-mapped cache has a better hit rate** than the set-associative cache. (Give the addresses as 8-digit binary numbers.) Also give the hit rate for each cache.

0000 0000
0010 0000
0110 0000
0000 0000

Direct Mapped: 25% hit rate
Set-Associative: 0% hit rate

2. [12 points] Caches & Branch Prediction

(a) [1 point] What is one advantage of a *write-back* cache over a *write-through* cache?

(i) Less bandwidth required, since less is written back

(b) [1 point] What is one advantage of a *write-through* cache over a *write-back* cache?

(i) No need for dirty bits
(ii) simpler

(c) [3 points] Describe a cache design (giving the total size, associativity and block size) that has a tag array that is *the same size* as the data array. Assume that addresses are 32 bits in size.

A direct mapped cache (4B) with a single 4B block
⇒ 30 bit tag
1 valid bit
1 dirty bit

(other answers possible)

(d) [2 points] What are the two purposes that a BTB serves?

i) -

ii) -

(e) [5 points] Consider a gshare branch direction predictor that has eight 2-bit saturating counters and a 2-bit history register. All counters are initialized to the strongly not-taken state (N), and the history is initialized to 00 (NN).

For each branch PC (given in binary) in the sequence below, give the predicted direction for each branch. Assume that these instructions come from a variable-length ISA where program counters are not necessarily aligned to any boundary.

branch PC	actual direction	predicted direction
1000 1101	T	-
0000 0100	T	-
0101 1001	N	-
1010 1111	T	-
0000 1100	T	-

5. [14 points] Caches

For the following questions, consider a system with 16-bit addresses, and a 2KB 2-way set-associative cache with 8B blocks and an LRU replacement policy.

(a) How many bits are in the tag, index and block offset for this cache?

#blocks = $\frac{2048B}{8B}$ = 256 (2-way) Index bits = $\log_2(128)$ = 7 bits

tag bits: 16 - 7 - 3 = 6 bits offset bits: $\log_2(8)$ = 3 bits

(b) What is the size of the tag array (in bytes) for this cache? Be sure to account for the valid and dirty bits, but you can ignore the LRU bit.

256 blocks in cache × (6 bit tag + 1 bit valid + 1 bit dirty)

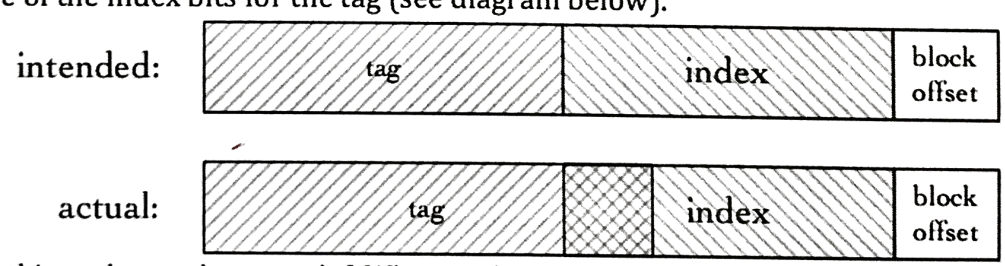
(256 × 8) bits

= 256 Bytes

(c) For an initially-empty cache, identify whether each address will hit or miss.

address	hit or miss?
0110 1100 0010 0010	miss
1100 1010 0010 0001	miss
0110 1100 0010 0010	hit
0110 1000 0010 0011	miss
1100 1100 0010 0000	miss
0110 1100 0010 0010	miss

(d) Dr. Chip D. Signer makes a mistake while designing a processor cache and reuses some of the index bits for the tag (see diagram below).



Will this cache work correctly? Why or why not?

(i) Yes, it will

(ii) It is just bigger than necessary (the tag i.e)

6. [22 points] Caches

For the following questions, consider a system with 16-bit addresses, and a 16KB direct-mapped cache with 32B blocks.

(a) How many bits are in the tag, index and block offset for this cache?

#blocks = $\frac{16KB}{32B}$ = 512 Blocks

Offset bits = $\log_2(32)$ = 5 bits
Index bits = $\log_2(512)$ = 9 bits
Tag bits = 16 - 9 - 5 = 2 bits

(b) What is the size of the tag array (in bytes) for this cache? Be sure to account for the valid and dirty bits.

512 blocks × (2 bit tag + 1 bit valid + 1 bit dirty)

512 × (4 bits)

256 Bytes

(c) For an initially-empty cache, identify whether each address will hit or miss.

address	hit or miss?
0100 1000 0110 0010	miss
1100 1010 1010 0001	miss
0100 1000 0110 1010	hit
0000 1000 1110 0000	miss
1100 1000 0111 0010	miss
0100 1000 0110 1011	miss

(d) Consider an alternate 16KB cache with 32B blocks but 2-way set associativity with an LRU replacement policy. How many bits are in this alternate cache's tag, index and block offset?

Tag bits: 3 bits
Index bits: 8 bits
Offset bits: 5 bits

(e) Give a sequence of **four** memory accesses that will have a **better hit rate in the direct-mapped cache** from part (a), than the set-associative cache from (d). All caches are initially empty. Specify the address of each access as a 16-bit binary number.

tag	index	offset	
001	0000 0000	00000	miss in both
010	0000 0000	00000	miss in both
110	0000 0000	00000	miss in both
001	0000 0000	00000	hit in DM, miss in SA

(f) Give a sequence of **four** memory accesses that will have a **better hit rate in the set-associative cache** from part (d), than the direct-mapped cache from (a). All caches are initially empty. Specify the address of each access as a 16-bit binary number.

000	0000	0000	00000	miss in both
100	0000	0000	00000	miss in both
000	0000	0000	00000	hit in SA, miss in DM
100	0000	0000	00000	hit in SA, miss in DM