NAME: _____

Computer Architecture and Engineering

CS152 Quiz #2

March 9th, 2010

Professor Krste Asanovic

Name:_____

This is a closed book, closed notes exam.

80 Minutes

9 Pages

Notes:

- **Not all questions are of equal difficulty, so look over the entire exam and budget your time carefully.**
- **Please carefully state any assumptions you make.**
- **Please write your name on every page in the quiz.**
- **You must not discuss a quiz's contents with students who have not yet taken the quiz. If you have inadvertently been exposed to the quiz prior to taking it, you must tell the instructor or TA.**
- **You will get no credit for selecting multiple-choice answers without giving explanations if the instructions ask you to explain your choice.**

Writing name on each sheet	**_____**	**1 Point**
Question 1	**_____**	**28 Points**
Question 2	**_____**	**28 Points**
Question 3	**_____**	**23 Points**
TOTAL	**_____**	**80 Points**

NAME: _____

Problem Q.1.A — Cache Performance [6 points]

This problem evaluates the cache performance of the following C code, which transposes a square matrix **A**, placing the result in another matrix **B**.

```
#define N 1024
int A[N*N], B[N*N];
int i,j;

for(i = 0; i < N; i++)
  for(j = 0; j < N; j++)
    B[j*N+i] = A[i*N+j];
```

Assume **A** and **B** are both aligned to a 4KB boundary and are contiguous in memory. **int**s are 32 bits (4 bytes).

Q.1.A – Cache Performance [6 points]
Consider a 4KB two-way set-associative cache with LRU replacement and 8-word (32-byte) cache lines. What will the store miss rate be when running the code? What will the load miss rate be when running the code?

This problem is simplified by LRU replacement: accesses to A and B will not conflict with each other in the cache. Thus, the store and load miss rates can be calculated independently of each other.

Loads are unit-stride. Every eighth load will cause a compulsory miss, and the intervening seven loads will hit. (Spatial locality is fully exploited.) The load miss rate is thus 1/8.

Stores have a 4KB stride. The first outer-loop iteration, all stores are compulsory misses. After that, all stores incur capacity misses before spatial locality can be exploited. The store miss rate is thus 100%.

NAME: _____

Problem Q.1.B-C

Cache Performance
[10 points]

Q.1.B – Replacement Policy [4 points]
How will the load miss rate change if a FIFO (first-in, first-out) replacement policy is employed instead of LRU?

In the vicinity of the matrix's diagonal, B's accesses will map to the same set as A's accesses. While LRU replacement kept A's accesses in the cache, FIFO will cause a cache miss on every second load in this situation. The result is three load conflict misses every outer loop iteration, resulting in a slightly higher load miss rate of (128+3)/1024.

Q.1.C – Cache Design [6 points]
Assuming the same cache parameters as in part A, which of the following cache configurations will perform best when executing this code? Circle one, and justify your answer for credit.

1) Write-Allocate, Write-Back
2) Write-Allocate, Write-Through (with a write buffer)
3) No Write-Allocate, Write-Back
4) No Write-Allocate, Write-Through (with a write buffer)

No Write-Allocate, Write-Through is the best configuration. Recall that all stores miss and there are no loads from the lines that are stored-to. Then, consider the amount of traffic between the L1 and L2 caches in each scenario. For every store, WA caches will cause 32 bytes of read traffic that is simply discarded. WA+WB caches will cause 32 bytes of write traffic on every store—a total of 64 bytes of L1<->L2 traffic for a 4 byte store. WA+WT caches reduce this to 36 bytes, but the allocation traffic is still wasteful.

NWA+WB caches behave similarly to NWA+WT: stores never hit, so writebacks never occur; thus, only 4 bytes are transferred to the L2 for every store in this code. But the lack of a write buffer makes the miss penalty much greater for NWA+WB than for NWA+WT, so the latter option is best.

NAME: _____

Problem Q.1.D-E

Cache/VM Performance [12 points]

Q.1.D – Code Optimization [7 points]
Now, consider a 4KB fully associative cache with LRU replacement and 8-word (32-byte) cache lines. Describe how the code could be rewritten to eliminate all non-compulsory misses.

An important insight is that matrix transposition, like matrix multiplication, can be blocked/tiled to take advantage of spatial locality for the stores. Since the cache is fully associative, several blocking strategies suffice; a simple one is to transpose 8x1 blocks at a time. This works because an entire cache line will be stored at a time, but associativity allows the 8 rows of A and the 1 row of B to be cached simultaneously, even though they map to the same set. (Note that this code requires that the block size B divides N).

```
#define N 1024
#define B 8
int A[N*N], B[N*N];
int i,j,k;

for(i = 0; i < N; i+=B)
  for(j = 0; j < N; j++)
    for(k = 0; k < B; k++)
      B[j*N+(i+k)] = A[(i+k)*N+j];
```

Credit was given to any strategy that eliminated non-compulsory misses without changing the semantics of the code. Source code was not required; a description of the blocking strategy was sufficient.

Q.1.E – Address Translation Performance [5 points]
Suppose the code is executed on a system with virtual memory. Assuming 4KB pages and a 1024-entry direct-mapped TLB, how many TLB misses occur when executing this code?

Each matrix is 4MB, so 2x4MB/4KB = 2048 compulsory misses occur. Let A_k refer to matrix A, row k (which occupies exactly one TLB entry). After some outer iteration i, $0 \le i < N-1$, TLB[k] contains B_k, $k \neq i$, and TLB[i] contains A_i. So, during outer iteration i+1, the store to B_i will miss, then the store to B_{i+1} will miss (since TLB[i+1] now contains A_{i+1}). Then, the next load to A_{i+1} will miss. So every iteration, 3 conflict misses occur (3072 total), for a total of 5120 including compulsory misses. (Actually, only one conflict occurs the zeroth iteration and two the last iteration, so the total is 5117, but this is a detail.)

Number of TLB Misses 5117

NAME: _____

Problem Q.2: Microtagged Cache

[28 points]

In this problem, we explore *microtagging*, a technique to reduce the access time of set-associative caches. Recall that for associative caches, the tag check must be completed before load results are returned to the CPU, because the result of the tag check determines which cache way is selected. Consequently, the tag check is often on the critical path.

The time to perform the tag check (and, thus, way selection) is determined in large part by the size of the tag. We can speed up way selection by checking only a subset of the tag—called a *microtag*—and using the results of this comparison to select the appropriate cache way. Of course, the full tag check must also occur to determine if the cache access is a hit or a miss, but this comparison proceeds in parallel with way selection. We store the full tags separately from the microtag array.

We will consider the impact of microtagging on a 4-way set-associative 16KB data cache with 32-byte lines. Addresses are 32 bits long. Microtags are 8 bits long. The baseline cache (i.e. without microtagging) is depicted in Figure H2-B in the attached handout. Figure 1, below, shows the modified tag comparison and driver hardware in the microtagged cache.

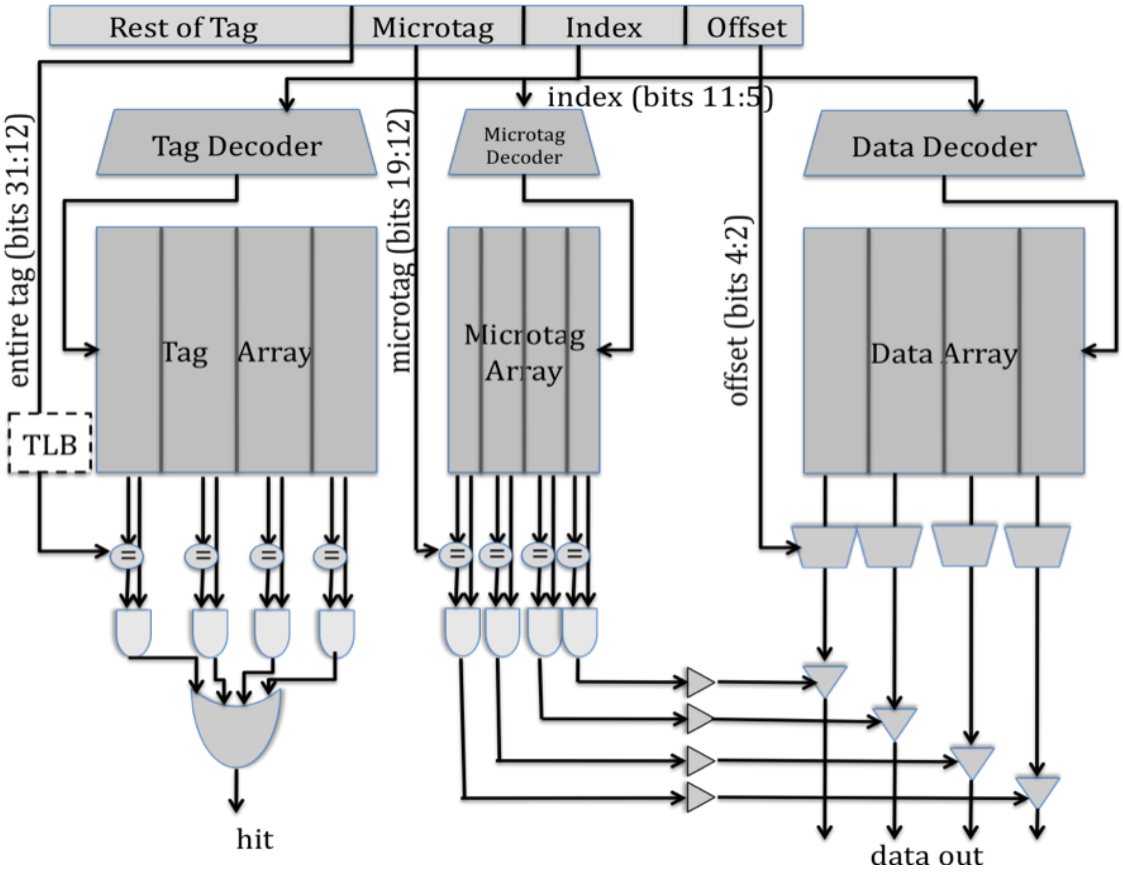

Figure 1: Microtagged cache datapath

NAME: _____

Problem Q.2.A

Cache Cycle Time
[8 points]

Q.2.A – Cycle Time [8 points]
Table 1, below, contains the delays of the components within the 4-way set-associative cache, for both the baseline and the microtagged cache. For both configurations, determine the critical path and the cache access time (i.e., the delay through the critical path).

Assume that the 2-input AND gates have a 50ps delay and the 4-input OR gate has a 100ps delay.

Component	Delay equation (ps)		Baseline	Microtagged
Decoder	20×(# of index bits) + 100	Tag	240	240
		Data	240	240
Memory array	20×$\log_2$ (# of rows) + 20×$\log_2$ (# of bits in a row) + 100	Tag	330	330
		Data	440	440
		Microtag		300
Comparator	20×(# of tag bits) + 100	Tag	500	500
		Microtag		260
N-to-1 MUX	50×$\log_2$ N + 100		200	200
Buffer driver	200		200	200
Data output driver	50×(associativity) + 100		300	300
Valid output driver	100		100	100

Table 1: Delay of each cache component

Old Critical Path _____

New Critical Path _____

Old Cycle Time _____ ps

New Cycle Time _____ ps

NAME: _____

Problem Q.2.B-C

Microtagged Cache
[10 points]

Q.2.B – AMAT [5 points]
Assume temporarily that both the baseline cache and the microtagged cache have the same hit rate, 95%, and the same average miss penalty, 20 ns. Using the cycle times computed in Q.1.A as the hit times, compute the average memory access time for both caches.

Old AMAT _____ ns

New AMAT _____ ns

Q.2.C – Microtagging Constraints [5 points]
Microtags add an additional constraint to the cache: in a given cache set, all microtags must be unique. This constraint is necessary to avoid multiple microtag matches in the same set, which would prevent the cache from selecting the correct way.

State which of the 3C's of cache misses this constraint affects. How will the cache miss rate compare to an ordinary 4-way set-associative cache? How will it compare to that of a direct-mapped cache of the same size?

NAME: _____

Problem Q.2.D-E

Virtual Memory
[10 points]

Q.2.D – Aliasing [5 points]
We now consider the effects of address translation on the microtagged cache. To avoid placing the TLB on the critical path, we use virtual microtags (i.e., the microtags are a subset of the virtual tag). The complete tags, however, are physical.

Without further modifications, it is possible for aliasing to occur in this cache. Carefully explain why this is the case.

Q.2.E – Anti-Aliasing [5 points]
Propose an efficient solution to prevent aliasing in this cache. (Hint: it is not necessary to use the technique discussed in lecture, i.e. using the L2 cache to detect aliases.)