Date:__________________

Quiz for Chapter 5 Large and Fast: Exploiting Memory Hierarchy

Not all questions are of equal difficulty. Please review the entire quiz first and then budget your time carefully.

Name:____

Course:____

Solutions in RED

1. [24 points] Caches and Address Translation. Consider a 64-byte cache with 8 byte blocks, an associativity of 2 and LRU block replacement. Virtual addresses are 16 bits. The cache is physically tagged. The processor has 16KB of physical memory.

(a) What is the total number of tag bits?

The cache is 64-bytes with 8-byte blocks, so there are 8 blocks. The associativity is 2, so there are 4 sets. Since there are 16KB of physical memory, a physical address is 14 bits long. Of these, 3 bits are taken for the offset (8-byte blocks), and 2 for the index (4 sets). That leaves 9 tag bits per block. Since there are 8 blocks, that makes 72 tag bits or 9 tag bytes.

(b) Assuming there are no special provisions for avoiding synonyms, what is the minimum page size?

To avoid synonyms, the number of sets times the block size cannot exceed the page size. Hence, the minimum page size is 32 bytes

(c) Assume each page is 64 bytes. How large would a single-level page table be given that each page requires 4 protection bits, and entries must be an integral number of bytes.

There is 16KB of physical memory and 64 B pages, meaning there are 256 physical pages and each PFN is 8 bits. Each entry has a PFN and 4 protection bits, that's 12 bits which we round up to 2 bytes. A single-level page table has one entry per virtual page. With a 16-bit virtual address space, there is 64KB of memory. Since each page is 64 bytes, that means there are 1K pages. At 2 bytes per page, the page table is 2KB in size.

(d) For the following sequence of references, label the cache misses.Also, label each miss as being either a compulsory miss, a capacity miss, or a conflict miss. The addresses are given in octal (each digit represents 3 bits). Assume the cache initially contains block addresses: 000, 010, 020, 030, 040, 050, 060, and 070 which were accessed in that order.

Cache state prior to access	Reference address	Miss ? Which ?
(00,04),(01,05),(02,06),(03,07)	024	
(00,04),(01,05),(06,02),(03,07)	100	
(04,10),(01,05),(06,02),(03,07)	270	
(04,10),(01,05),(06,02),(07,27)	570	
(04,10),(01,05),(06,02),(27,57)	074	
(04,10),(01,05),(06,02),(57,07)	272	
(04,10),(01,05),(06,02),(07,27)	004	
(10,00),(01,05),(06,02),(07,27)	044	

(00,04),(01,05),(06,02),(07,27)	640	
(04,64),(01,05),(06,02),(07,27)	000	
(64,00),(01,05),(06,02),(07,27)	410	
(64,00),(05,41),(06,02),(07,27)	710	
(64,00),(41,71),(06,02),(07,27)	550	
(64,00),(71,55),(06,02),(07,27)	570	
(64,00),(71,55),(06,02),(27,57)	410	

Since there are 8-byte blocks and addresses are in octal, to keep track of blocks you need only worry about the two most significant digits in each address, so 020 and 024 are in the same block, which we write as 02. Now, the cache is arranged into 4 sets of 2 blocks each. To map a block to a set, use (address / blocksize) % #sets, where address / blocksize is basically the address minus its last octal digit. Block 024 is in set (02) % 4 which is set #2. In short, set mappings are determined by the middle digit of the address. 0 and 4 go into set #0, 1 and 5 into set #1, 2 and 6 into set #2, and 3 and 7 into set #3. From there, just run LRU on each of the individual sets.

Now, to separate the misses, any miss to a block you have not seen before is a compulsory miss. Any miss to a block you have not seen within the last N distinct blocks where N is the total number of blocks in the cache is a capacity miss. (Basically, the last N distinct blocks will be the N blocks present in a fully associative cache). All other misses are conflict misses.

Cache state prior to access	Reference address	Miss ? Which ?
(00,04),(01,05),(02,06),(03,07)	024	Hit (update 02 LRU)
(00,04),(01,05),(06,02),(03,07)	100	Compulsory miss
(04,10),(01,05),(06,02),(03,07)	270	Compulsory miss
(04,10),(01,05),(06,02),(07,27)	570	Compulsory miss
(04,10),(01,05),(06,02),(27,57)	074	Conflict miss
(04,10),(01,05),(06,02),(57,07)	272	Conflict miss
(04,10),(01,05),(06,02),(07,27)	004	Capacity miss
(10,00),(01,05),(06,02),(07,27)	044	Capacity miss
(00,04),(01,05),(06,02),(07,27)	640	Compulsory miss
(04,64),(01,05),(06,02),(07,27)	000	Conflict miss
(64,00),(01,05),(06,02),(07,27)	410	Compulsory miss
(64,00),(05,41),(06,02),(07,27)	710	Compulsory miss
(64,00),(41,71),(06,02),(07,27)	550	Compulsory miss
(64,00),(71,55),(06,02),(07,27)	570	Capacity miss
(64,00),(71,55),(06,02),(27,57)	410	Conflict miss

(e) Which of the following techniques are aimed at reducing the cost of a miss: dividing the current block into sub-blocks, a larger block size, the addition of a second level cache, the addition of a victim buffer, early restart with critical word first, a writeback buffer, skewed associativity, software prefetching, the use of a TLB, and multi-porting.

The techniques that reduce miss cost are sub-blocking (allows smaller blocks to be transferred on a miss), the addition of a second level cache (obviously), early restart/critical word first (allows a miss to return to the processor earlier), and a write-back buffer (buffers write-backs allowing replacement blocks to access the bus immediately).

A larger block size (without early restart or sub-blocking) actually increases miss cost. Skewed associativity and software prefetching, reduce the miss rate, not the cost of a miss. A TLB reduces the cost of a hit (over the use of a serial TB) and is in general a functionality structure rather than a performance structure. Multi-porting increases cache bandwidth.

A victim buffer can reduce miss cost or not, depending on your definition. If you count the victim buffer as part of the cache (which it typically is), then it reduces miss rate, not miss cost. If you count it on its own, or as part of the next level of cache (which it really isn't, but OK), then it reduces miss cost.

(f) Why are the first level caches usually split (instructions and data are in different caches) while the L2 is usually unified (instructions and data are both in the same cache)?

The primary reason for splitting the first level cache is bandwidth, i.e., to avoid structural hazards. A typical processor must access instructions and data in the same cycle quite frequently. Supplying this kind of bandwidth in a unified cache requires either replication (in which case you may as well have a split cache) or multi-porting or multi-banking, both of which would slow down the hit time. Structural hazards are a much smaller issue for a second-level cache which is accessed much less frequently. Here the primary design goal of a second level cache is a low miss-rate. By combining instructions and data in the same cache, the capacity of the cache will be better utilized and a lower overall miss rate may be achieved.

2. [6 points] A two-part question.

(Part A)

Assume the following 10-bit address sequence generated by the microprocessor:

Time	0	1	2	3	4	5	6	7
Access	10001101	10110010	10111111	10001100	10011100	11101001	11111110	11101001
TAG								
SET								
INDEX								

The cache uses 4 bytes per block. Assume a 2-way set assocative cache design that uses the LRU algorithm (with a cache that can hold a total of 4 blocks). Assume that the cache is initially empty. First determine the TAG, SET, BYTE OFFSET fields and fill in the table above. In the figure below, clearly mark for each access the TAG, Least Recently Used (LRU), and HIT/MISS information for each access.

Initial

	Block 0		Block 1
Set 0			
Set 1			

Access 0

	Block 0		Block 1
Set 0			
Set 1			

Access 1

	Block 0		Block 1
Set 0			
Set 1			

Name:______________________________________

Access 2			
	Block 0		Block 1
Set 0			
Set 1			

Access 3			
	Block 0		Block 1
Set 0			
Set 1			

Access 4			
	Block 0		Block 1
Set 0			
Set 1			

Access 5			
	Block 0		Block 1
Set 0			
Set 1			

Access 6			
	Block 0		Block 1
Set 0			
Set 1			

Access 7			
	Block 0		Block 1
Set 0			
Set 1			

Time	0	1	2	3	4	5	6	7
Access	10001101	10110010	10111111	10001100	10011100	11101001	11111110	11101001
TAG	10001	10110	10111	10001	10011	11101	11111	11101
SET	1	0	1	1	1	0	1	0
INDEX	01	10	11	00	00	01	10	01

In the following each access is represented by a triple: (Tag, LRU Bit, Hit Bit)

LRU bit = 1 if the current block is the least recently used one.
Hit Bit = 1 if the current reference is a hit

Initial			
	Block 0		Block 1
Set 0			
Set 1			

Access 0			
	Block 0		Block 1
Set 0			
Set 1	10001,0,0		

Access 1			
	Block 0		Block 1
Set 0	10110,0,0		
Set 1	10001,0,0		

Name:____

Access 2		
	Block 0	Block 1
Set 0	10110,0,0	
Set 1	10001,1,0	10111,0,0

Access 3		
	Block 0	Block 1
Set 0	10110,0,0	
Set 1	10001,0,1	10111,1,0

Access 4		
	Block 0	Block 1
Set 0	10110,0,0	
Set 1	10001,1,0	10011,0,0

Access 5		
	Block 0	Block 1
Set 0	10110,1,0	11101,0,0
Set 1	10001,1,0	10011,0,0

Access 6		
	Block 0	Block 1
Set 0	10110,1,0	11101,0,0
Set 1	11111,0,0	10011,1,0

Access 7		
	Block 0	Block 1
Set 0	11101,0,0	11101,0,0
Set 1	11111,0,0	10011,1,0

(Part B)

Derive the hit ratio for the access sequence in Part A.

The hit ratio for the above access sequence is given by : (1/8) = 0.125

3. [6 points] A two part question

(a) Why is miss rate not a good metric for evaluating cache performance? What is the appropriate metric? Give its definition. What is the reason for using a combination of first and second- level caches rather than using the same chip area for a larger first-level cache?

The ultimate metric for cache performance is average access time: tavg = thit + miss-rate * tmiss. The miss rate is only one component of this equation. A cache may have a low miss rate, but an extremely high penalty per miss, making it lower-performing than a cache with a higher miss rate but a substantially lower miss penalty. Alternatively, it may have a low miss rate but a high hit time (this is true for large fully associative caches, for instance).

Multiple levels of cache are used for exactly this reason. Not all of the performance factors can be optimized in a single cache. Specifically, with tmiss (memory latency) given, it is difficult to design a cache which is both fast in the common case (a hit) and minimizes the costly uncommon case by having a low miss rate. These two design goals are achieved using two caches. The first level cache minimizes the hit time, therefore it is usually small with a low-associativity. The second level cache minimizes the miss rate, it is usually large with large blocks and a higher associativity.

(b) The original motivation for using virtual memory was "compatibility". What does that mean in this context? What are two other motivations for using virtual memory?

Compatibility in this context means the ability to transparently run the same piece of (un-recompiled) software on different machines with different amounts of physical memory. This compatibility freed the application writer from worrying about how much physical memory a machine had.

The motivation for using virtual memory today have mainly to do with multiprogramming, the ability to interleave the execution of multiple programs simultaneously on one processor. Virtual memory provides protection, relocation, fast startup, sharing and communication, and the ability to memory map peripheral devices.

4. [6 points] A four part question

(Part A)

What are the two characteristics of program memory accesses that caches exploit?

Temporal and spatial locality

(Part B)

What are three types of cache misses?

Cold misses, conflict misses and compulsory misses

(Part C)

Design a 128KB direct-mapped data cache that uses a 32-bit address and 16 bytes per block. Calculate the following:

(a) How many bits are used for the byte offset?

7 bits

(b) How many bits are used for the set (index) field?

15 bits

(c) How many bits are used for the tag?

10 bits

(Part D)

Design a 8-way set associative cache that has 16 blocks and 32 bytes per block. Assume a 32 bit address. Calculate the following:

(a) How many bits are used for the byte offset?

5 bits

(b) How many bits are used for the set (index) field?

2 bits

(c) How many bits are used for the tag?

25 bits

5. [6 points] The memory architecture of a machine X is summarized in the following table.

Virtual Address	54 bits
Page Size	16 K bytes
PTE Size	4 bytes

(a) Assume that there are 8 bits reserved for the operating system functions (protection, replacement, valid, modified, and Hit/Miss- All overhead bits) other than required by the hardware translation algorithm. Derive the largest physical memory size (in bytes) allowed by this PTE format. Make sure you consider all the fields required by the translation algorithm.

Since each Page Table element has 4 bytes (32 bits) and the page size is 16K bytes:

(1) we need $\log_2(16*2^{10}*2^3)$, ie, 17 bits to hold the page offset

(2) we need 32 – 8 (used for protection) = 24 bits for page number

The largest physical memory size is $2^{(17+24)}/2^{(3)}$ bytes = 2^{38} bytes = 256 GB

(b) How large (in bytes) is the page table?

The page table is indexed by the virtual page number which uses 54 – 14 = 40 bits. The number of entries are therefore 2^{40}. Each PTE has 4 bytes. So the total size of the page table is 2^{42} bytes which is 4 terabytes.

(c) Assuming 1 application exists in the system and the maximum physical memory is devoted to the process, how much physical space (in bytes) is there for the application's data and code.

The application's page table has an entry for every physical page that exists on the system, which means the page table size is $2^{24}*4$ bytes. This leaves the remaining physical space to the process:

$2^{38} - 2^{26}$ bytes

6. [6 points] This question covers virtual memory access. Assume a 5-bit virtual address and a memory system that uses 4 bytes per page. The physical memory has 16 bytes (four page frames). The page table used is a one-level scheme that can be found in memory at the PTBR location. Initially the table indicates that no virtual pages have been mapped. Implementing a LRU page replacement algorithm, show the contents of physical memory after the following virtual accesses: 10100, 01000, 00011, 01011, 01011,11111. Show the contents of memory and the page table information after each access successfully completes in figures that follow. Also indicate when a page fault occurs. Each page table entry (PTE) is 1 byte.