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	

Name:_____

(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.

Name:_____

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.

Name:_____

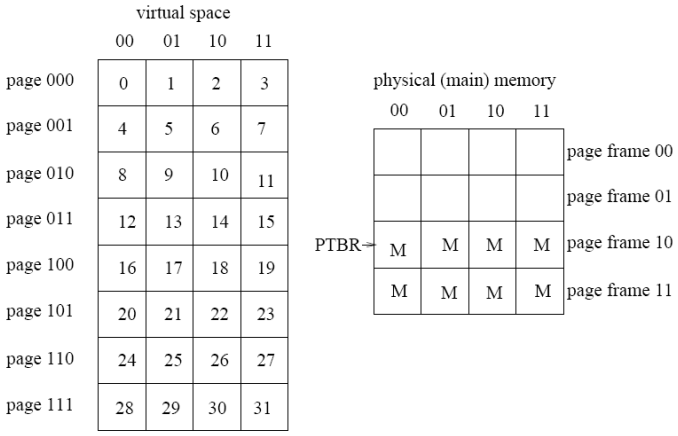


Figure 1: The initial contents of memory.

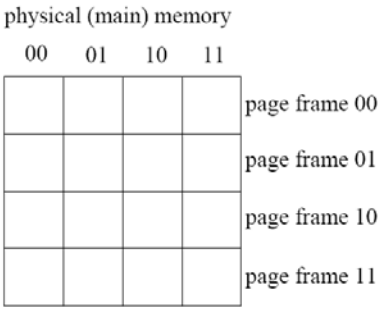


Figure 2: Figure (after access 10100).

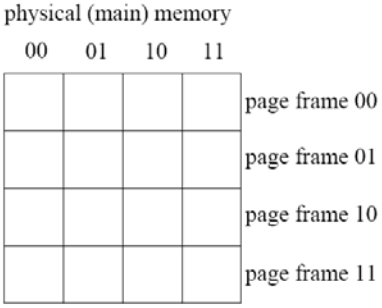


Figure 3: Figure (after access 01000).

Name:_____

physical (main) memory

00	01	10	11	
				page frame 00
				page frame 01
				page frame 10
				page frame 11

Figure 4: Figure (after access 00011).

physical (main) memory

00	01	10	11	
				page frame 00
				page frame 01
				page frame 10
				page frame 11

Figure 5: Figure (after access 01011).

physical (main) memory

00	01	10	11	
				page frame 00
				page frame 01
				page frame 10
				page frame 11

Figure 6: Figure (after access 01011).

physical (main) memory

00	01	10	11	
				page frame 00
				page frame 01
				page frame 10
				page frame 11

Figure 7: Figure (after access 11111).

Name:_____

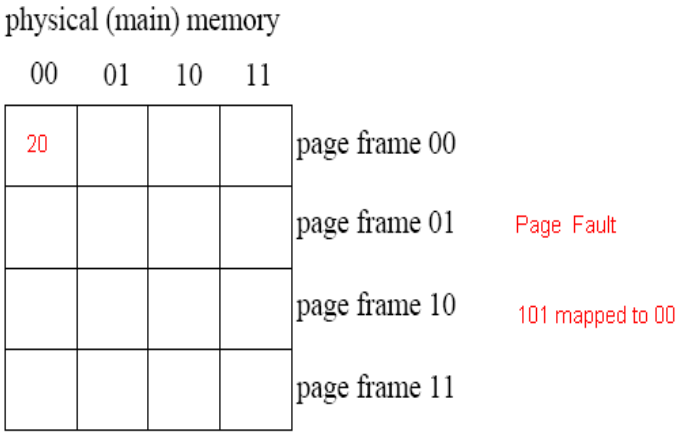


Figure 2: Figure (after access 10100).

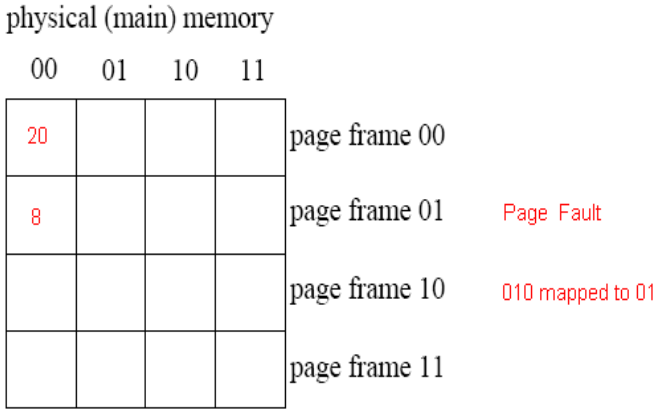


Figure 2: Figure (after access 10100).

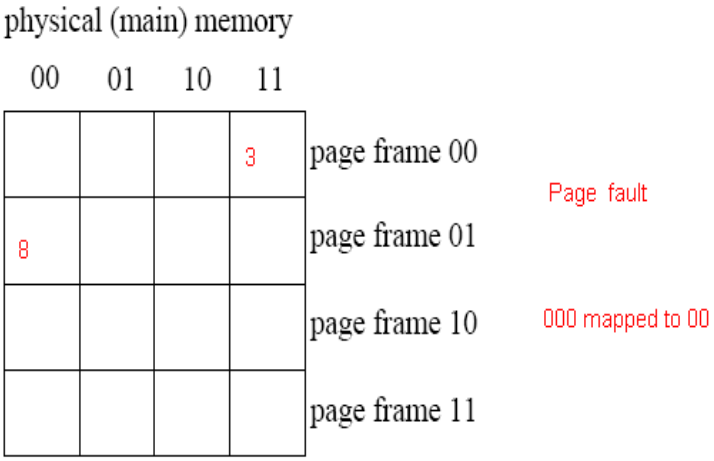


Figure 4: Figure (after access 00011).

Name:_____

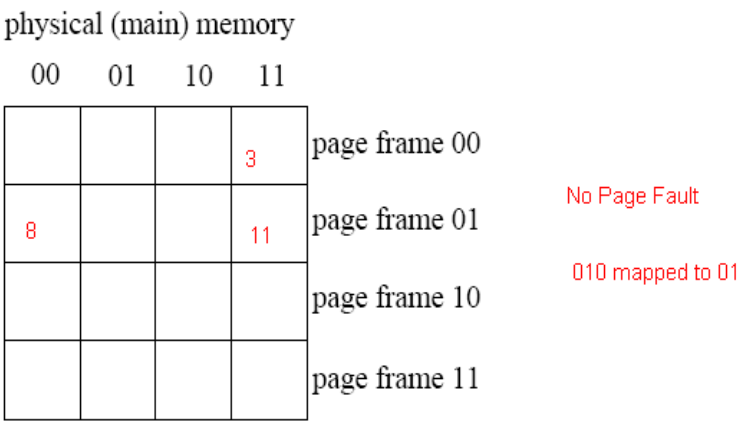


Figure 5: Figure (after access 01011).

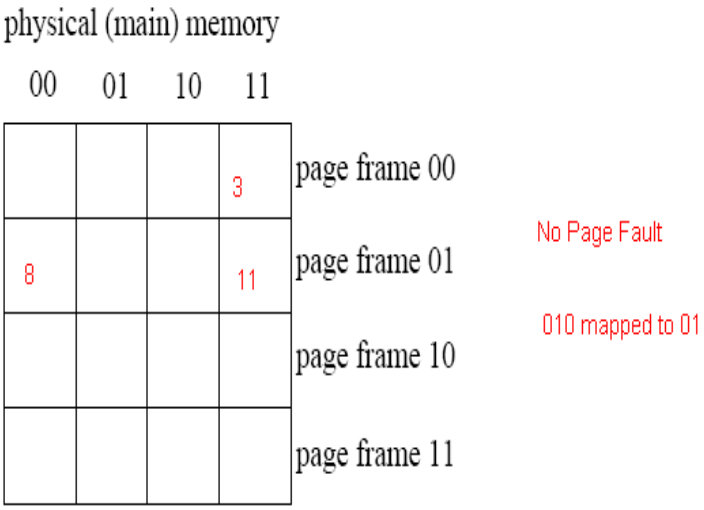


Figure 5: Figure (after access 01011).

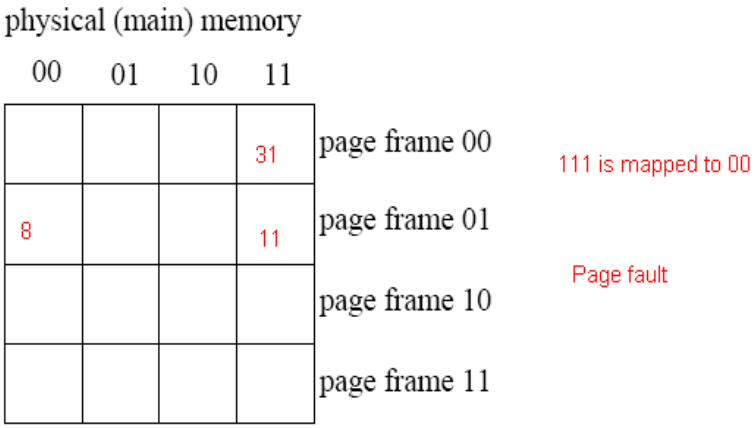


Figure 7: Figure (after access 11111).

Name:_____

7. [6 points] A multipart question.

(Part A)

In what pipeline stage is the branch target buffer checked?

(a) Fetch
(b) Decode
(c) Execute
(d) Resolve

The branch target buffer is checked at (a) Fetch stage.

What needs to be stored in a branch target buffer in order to eliminate the branch penalty for an unconditional branch?

(a) Address of branch target
(b) Address of branch target and branch prediction
(c) Instruction at branch target.

To eliminate the branch penalty the BTB needs to store (b) Address of branch target and branch prediction

The Average Memory Access Time equation (AMAT) has three components: hit time, miss rate, and miss penalty. For each of the following cache optimizations, indicate which component of the AMAT equation is improved.

- Using a second-level cache
- Using a direct-mapped cache
- Using a 4-way set-associative cache
- Using a virtually-addressed cache
- Performing hardware pre-fetching using stream buffers
- Using a non-blocking cache
- Using larger blocks

- Using a second-level cache improves miss rate
- Using a direct-mapped cache improves hit time
- Using a 4-way set-associative cache improves miss rate
- Using a virtually-addressed cache improves hit time
- Performing hardware pre-fetching using stream buffers improves miss rate
- Using a non-blocking cache improves miss penalty
- Using larger blocks improves miss rate

Name:_____

(Part B)

(a) (True/False) A virtual cache access time is always faster than that of a physical cache?

True (for a single level cache) since the virtual to physical address translation is avoided.

(b) (True/False) High associativity in a cache reduces compulsory misses.

False. It reduces conflict misses.

(c) (True/False) Both DRAM and SRAM must be refreshed periodically using a dummy read/write operation.

False. only DRAM's need to be refreshed periodically.

(d) (True/False) A write-through cache typically requires less bus bandwidth than a write-back cache.

False. It is the other way round

(e) (True/False) Cache performance is of less importance in faster processors because the processor speed compensates for the high memory access time.

False. The huge processor-memory gap implies cache performance is very critical.

(f) (True/False) Memory interleaving is a technique for reducing memory access time through increased bandwidth utilization of the data bus.

True.

8. [12 points] A three-part question. This question covers cache and pipeline performance analysis.

(Part A)

Write the formula for the average memory access time assuming one level of cache memory:

Average Memory Access Time = Time for a hit + Miss rate × Miss penalty

(Part B)

For a data cache with a 92% hit rate and a 2-cycle hit latency, calculate the average memory access latency. Assume that latency to memory and the cache miss penalty together is 124 cycles. Note: The cache must be accessed after memory returns the data.

AMAT = 2 + 0.08*124
= 11.92 cycles

Name:_____

(Part C)

Calculate the performance of a processor taking into account stalls due to data cache and instruction cache misses. The data cache (for loads and stores) is the same as described in Part B and 30% of instructions are loads and stores. The instruction cache has a hit rate of 90% with a miss penalty of 50 cycles. Assume the base CPI using a perfect memory system is 1.0. Calculate the CPI of the pipeline, assuming everything else is working perfectly. Assume the load never stalls a dependent instruction and assume the processor must wait for stores to finish when they miss the cache. Finally, assume that instruction cache misses and data cache misses never occur at the same time. Show your work.

- Calculate the additional CPI due to the icache stalls.
- Calculate the additional CPI due to the dcache stalls.
- Calculate the overall CPI for the machine.

The additional CPI due to icache stalls = Hit Rate * Hit Latency + Miss Rate* Miss Penalty
= 0.9*2 + 0.1*50
= 1.8 + 5
= 6.8

The additional CPI due to dcache stalls = 0.92*2 + 0.08*124
= 11.76

The overall CPI = 0.3*11.76 + 0.7*1.0 + 1.0*6.8
= 11.03

9. [12 points] A three-part question.

(Part A)

A processor has a 32 byte memory and an 8 byte direct-mapped cache. Table 0 shows the current state of the cache. Write hit or miss under the each address in the memory reference sequence below. Show the new state of the cache for each miss in a new table, label the table with the address, and circle the change:

Addr	10011	00001	00110	01010	01110	11001	00001	11100	10100
H/M									

Name:_____

0. Initial state

Index	V	Tag	Data
000	N		
001	Y	00	Mem(00001)
010	N		
011	Y	11	Mem(11011)
100	Y	10	Mem(10100)
101	Y	01	Mem(01101)
110	Y	00	Mem(00110)
111	N		

1.

Index	V	Tag	Data
000			
001			
010			
011			
100			
101			
110			
111			

2.

Index	V	Tag	Data
000			
001			
010			
011			
100			
101			
110			
111			

3.

Index	V	Tag	Data
000			
001			
010			
011			
100			
101			
110			
111			

4.

Index	V	Tag	Data
000			
001			
010			
011			
100			
101			
110			
111			

5.

Index	V	Tag	Data
000			
001			
010			
011			
100			
101			
110			
111			

6.

Index	V	Tag	Data
000			
001			
010			
011			
100			
101			
110			
111			

7.

Index	V	Tag	Data
000			
001			
010			
011			
100			
101			
110			
111			

Name:_____

Addr	10011	00001	00110	01010	01110	11001	00001	11100	10100
H/M	M	H	H	M	M	M	M	M	M

0. Initial state

Index	V	Tag	Data
000	N		
001	Y	00	Mem(00001)
010	N		
011	Y	11	Mem(11011)
100	Y	10	Mem(10100)
101	Y	01	Mem(01101)
110	Y	00	Mem(00110)
111	N		

1. 10011

Index	V	Tag	Data
000	N		
001	Y	00	m[00001]
010	N		
011	Y	10	m[10011] ←
100	Y	10	m[10100]
101	Y	01	m[01101]
110	Y	00	m[00110]
111	N		

2. 01010

Index	V	Tag	Data
000	N		
001	Y	00	m[00001]
010	Y	01	m[01010] ←
011	Y	10	m[10011]
100	Y	10	m[10100]
101	Y	01	m[01101]
110	Y	00	m[00110]
111	N		

3. 01110

Index	V	Tag	Data
000	N		
001	Y	00	m[00001]
010	Y	01	m[01010]
011	Y	10	m[10011]
100	Y	10	m[10100]
101	Y	01	m[01101]
110	Y	01	m[01110] ←
111	N		

4. 11001

Index	V	Tag	Data
000	N		
001	Y	11	m[11001] ←
010	Y	01	m[01010]
011	Y	10	m[10011]
100	Y	10	m[10100]
101	Y	01	m[01101]
110	Y	01	m[01110]
111	N		

5. 00001

Index	V	Tag	Data
000	N		
001	Y	00	m[00001] ←
010	Y	01	m[01010]
011	Y	10	m[10011]
100	Y	10	m[10100]
101	Y	01	m[01101]
110	Y	01	m[01110]
111	N		

6. 11100

Index	V	Tag	Data
000	N		
001	Y	00	m[00001]
010	Y	01	m[01010]
011	Y	10	m[10011]
100	Y	11	m[11100] ←
101	Y	01	m[01101]
110	Y	01	m[01110]
111	N		

7.

Index	V	Tag	Data
000	N		
001	Y	00	m[00001]
010	Y	01	m[01010]
011	Y	10	m[10011]
100	Y	10	m[10100] ←
101	Y	01	m[01101]
110	Y	01	m[01110]
111	N		

Name:_____

(Part B)

Do the same thing as in Part A, except for a 4-way set associative cache. Assume 00110 and 11011 were the last two addresses to be accessed. Use the Least Recently Used replacement policy.

Addr	10011	00001	00110	01010	01110	11001	00001	11100	10100
H/M									

0.

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0			0011	Mem(00110)			1010	Mem(10100)
1	0000	Mem(00001)	1101	Mem(11011)	0110	Mem(01101)		

1.

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

2.

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

3.

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

4.

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

5.

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

6.

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

7.

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Name:_____

Addr	10011	00001	00110	01010	01110	11001	00001	11100	10100
H/M	M	H	H	M	M	M	H	M	M

0. initial state

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0			0011	Mem(00110)			1010	Mem(10100)
1	0000	Mem(00001)	1101	Mem(11011)	0110	Mem(01101)		

1. 10011

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0			0011	m[00110]			1010	m[10100]
1	0000	m[00001]	1101	m[11011]	0110	m[01101]	1001	m[10011]

2. 01010

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0	0101	m[01010]	0011	m[00110]			1010	m[10100]
1	0000	m[00001]	1101	m[11011]	0010	m[01101]	1001	m[10011]

3. 01110

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0	0101	m[01010]	0011	m[00110]	0111	m[01110]	1010	m[10100]
1	0000	m[00001]	1101	m[11011]	0110	m[01101]	1001	m[10011]

4. 11001

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0	0101	m[01010]	0011	m[00110]	0111	m[01110]	1010	m[10100]
1	0000	m[00001]	1101	m[11011]	1100	m[11001]	1001	m[10011]

5. 11100

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0	0101	m[01010]	0011	m[00110]	0111	m[01110]	1110	m[11100]
1	0000	m[00001]	1101	m[11011]	1100	m[11001]	1001	m[10011]

6. 10100

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0	0101	m[01010]	1010	m[10100]	0111	m[01110]	1110	m[11100]
1	0000	m[00001]	1101	m[11011]	1100	m[11001]	1001	m[10011]

7.

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

(Part C)

(a) What is the hit and miss rate of the direct-mapped cache in Part A?

Hit Rate = 2/9
Miss Rate = 7/9

Name:_____

(b) What is the hit and miss rate of the 4-way set associative cache in Part B?

Hit Rate = 3/9
Miss Rate = 6/9

(c) Assume a machine with a CPI of 4 and a miss penalty of 10 cycles. Ignoring writes, calculate the ratio of the performance of the 4-way set associative cache to the direct-mapped cache. In other words, what is the speedup when using the machine with the 4-way cache?

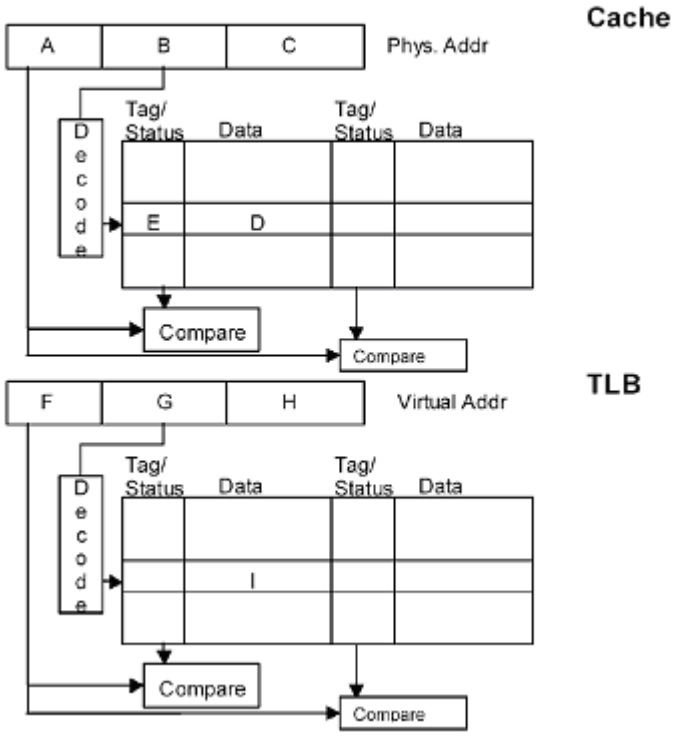
Number of Hits = 4
Number of Misses = 14

SpeedUp = ((4*2)/9 + (7*14)/9)/((4*3/9) +(6*14/9)) = 1.105

10. [6 points] Consider a memory system with the following parameters:

- Translation Lookaside Buffer has 512 entries and is 2-way set associative.
- 64Kbyte L1 Data Cache has 128 byte lines and is also 2-way set associative.
- Virtual addresses are 64-bits and physical addresses are 32 bits.
- 8KB page size

Below are diagrams of the cache and TLB. Please fill in the appropriate information in the table that follows the diagrams:



Name:_____

L1 Cache		TLB	
A =	bits	F =	Bits
B =	bits	G =	Bits
C =	bits	H =	Bits
D =	bits	I =	Bits
E =	bits		

L1 Cache		TLB	
A =	17 bits	F =	43 Bits
B =	8 bits	G =	8 Bits
C =	7 bits	H =	13 Bits
D =	512 bits	I =	19 Bits
E =	19 bits		

11. [3 points] Invalidation vs. Update-based Protocols.

(a) As miss latencies increase, does an update protocol become more or less preferable to an invalidation-based protocol? Explain.

As miss latencies increase, an update protocol would be more preferable to an invalidation protocol. This is because an invalidation protocol just flushes a cache line on a processor whenever some other processor writes to a cache line which has the same mapped block and hence a subsequent load (which may occur much later after the invalidation) will incur a higher miss penalty with a higher miss latency. In case of an update based protocol, since the other processor updates the cache line as and when it writes any value, the miss latency is not the critical determining factor of the subsequent load.

(b) In a multilevel cache hierarchy, would you propagate updates all the way to the first-level cache or only to the second-level cache? Explain the trade-offs.

Propagating updates all the way to the first cache could result in unnecessary traffic being thrust upon the first level cache of one processor just because it has the memory blocks being mapped into its cache the first time it was accessed and was not evicted since then. This could increase the hit time. On the other hand this could also result in tolerating L1 miss latency to a greater extent than before if the blocks that are updated are immediately being used up by the second processor, which is especially true for shared memory workloads.

12. [6 points] How many total SRAM bits will be required to implement a 256KB four-way set associative cache. The cache is physically-indexed cache, and has 64-byte blocks. Assume that there are 4 extra bits per entry: 1 valid bit, 1 dirty bit, and 2 LRU bits for the replacement policy. Assume that the physical address is 50 bits wide.

The number of sets in the 256KB four-way set associative cache = $(256*2^{10})/(4*64)$ =1024

A set has four entries. Each entry in the set occupies 4 bits + 64*8 bits = 516 bits

The total number of SRAM bits required = (516)*4*1024 = 2113536

Name:_____

13. [6 points] TLB's are typically built to be fully-associative or highly set-associative. In contrast, first-level data caches are more likely to be direct-mapped or 2 or 4-way set associative. Give two good reasons why this is so.

TLB's cache the translation from a virtual page number to a physical page number. Since
(a) a page size is much larger than a normal cache line size, there is likely to very less spatial locality amongst the virtual address accesses from a single process than between successive memory blocks.

(b) Different processes have their own virtual address space and hence to prevent a lot of conflict misses to ensure TLBs don't become a source of unfairness as far as a process's memory access is concerned, they are designed to be fully associative.

14. [6 points] Caches: Misses and Hits

```
int i;
int a[1024*1024];
int x=0;

for(i=0;i<1024;i++)
{
x+=a[i]+a[1024*i];
}
```

Consider the code snippet in code above. Suppose that it is executed on a system with a 2-way set associative 16KB data cache with 32-byte blocks, 32-bit words, and an LRU replacement policy. Assume that int is word-sized. Also assume that the address of a is 0x0, that i and x are in registers, and that the cache is initially empty. How many data cache misses are there? How many hits are there?

The number of sets in the cache = $(16 * 2^{10}) / (2*32) = 256$

Since a word size is 4 bytes, int is word sized and the size of a cache block is 32 bytes, the number of ints that would fit in a cache block is 8.

Therefore all the ints in a from a[0] to a[1023] map to the one of the cache lines of the sets 0 to 127, while all the ints in a from a[1024] to a[1024*2 -1] map to the sets 128 to 255. Similarly the array elements a[1024*2] to a[1024*3-1] map to cache lines of sets 0 to 127, a[1024*3] to a[1024*4 – 1] map to cache lines 128 to 255 and so on.

In the loop, every time a[i] is accessed for i being a multiple of 8 would be a miss. There the number of misses due to a[i] accesses inside the loop is 1024/8 = 128. The number of hits is 512.

Now all accesses to a[1024*i] within the loop are misses. This is because map alternately to sets 0 and 128 consecutively where there are cold misses the first time they are referenced.

The total number of misses = 1024 + 128 = 1062

The total number of hits = 512.

Name:_____

15. [6 points] Virtual Memory

(a) 32-bit Virtual Address Spaces. Consider a machine with 32-bit virtual addresses, 32-bit physical addresses, and a 4KB page size. Consider a two-level page table system where each table occupies one full page. Assume each page table entry is 32 bits long. To map the full virtual address space, how much memory will be used by the page tables?

The number of bits for the page offset = 12 bits

Since a page table occupies a page and each page table entry is 32-bits long, the number of entries in the page table = 4KB/32 = 1024

10 bits of the virtual address are used to index the top level page table. 10 more bits are needed to index the rest of the 1024 tables.

Amount of memory used = 4 KB + 1024*(4 KB) = 2052 KB

(b) 64-bit Virtual Address Spaces. A two-part question.

(Part 1)

Consider a machine with a 64-bit virtual addresses, 64-bit physical addresses, and a 4MB page size. Consider a two-level page table system where each table occupies one full page. Assume each page table entry is 64 bits long. To map the full virtual address space, how much memory will be used by the page tables? (Hint: you will need more than 1 top-level page table. For this question this is okay.)

The number of bits for the page offset = 22 bits

Since a page table occupies a page and each page table entry is 64-bits long, the number of entries in the page table = 4MB/64 = 65,636

16 bits of the virtual address is used to index the second level page tables.

Of the remaining bits are 26 bits, 10 bits are used to find 1024 different top level tables while 16 bits are used to index one of these top level tables.

Total Memory used up = 1024*4MB + 65,536*4 MB = 260 GB

(Part 2)

Rather than a two-level page table, what other page table architecture could be used to reduce the memory foot print of page tables for the 64-bit address space from the last question? Assume that you do not need to map the full address space, but some small fraction (people typically do not have 264 bytes of physical memory). However, you should assume that the virtual pages that are mapped are uniformly distributed across the virtual address space (i.e. it is not only the low addresses or high addresses that are mapped, but rather a few pages from all ranges of memory).

One can add one more level of page table hierarchy having a page directory, and then two levels of page tables, as is done in operating systems like Linux. Another implementation could be by using hashing, a technique used to maintain an efficient data structure for accessing and storing only a few items from a domain of different elements.

Name:_____

16. [6 points] Consider an architecture that uses virtual memory, a two-level page table for address translation, as well as a TLB to speed up address translations. Further assume that this machine uses caches to speed up memory accesses. Recall that all addresses used by a program are virtual addresses. Further recall that main memory in the microarchitecture is indexed using physical addresses. The virtual memory subsystem and cache memories could interact in several ways. In particular, the cache memories could be accessed using virtual addresses. We will refer to this scheme as a virtually indexed, virtually tagged cache. The cache could be indexed using virtual addresses, but the tag compare could happen with physical addresses (virtually indexed, physically tagged). Finally, the cache could be accessed using only the physical address. Describe the virtues and drawbacks for each of these systems. Be sure to consider the case where two virtual addresses map to the same physical address.

	Virtually Indexed, Virtually Tagged	Virtually indexed, physically tagged	Physically indexed, physically tagged
Advantages	Translation of virtual to physical address is required only on a miss	Performance is good	(a) Allows multiple processes to have blocks in cache at same time, share pages, etc. (b) Access rights checked as part of translation
Disadvantages	Two processes having the same virtual address map to different physical address and so extra bits needed to distinguish between processes using some kind of ids	Requires cache index to remain invariant across translation	Not fast compared to the other two as have to wait for virtual to physical address translation

17. [6 points] Describe the general characteristics of a program that would exhibit very little temporal and spatial locality with regard to instruction fetches. Provide an example of such a program (pseudo-code is fine). Also, describe the cache effects of excessive unrolling. Use the terms static instructions and dynamic instructions in your description.

Any code that generates a lot of indirect jumps to functions that are separated quite far apart in terms of their layout in memory is likely to exhibit very little temporal and spatial locality. An example is an interpreter for some language that maintains a table of function pointers and when fed a program jumps to these functions based on the code in the interpreted program is an example of this behavior.

Example pseudo-code:

```
function_pointer_t table[] = { handle_jump, handle_print, handle_add,
,..., handle_halt }

int main()
{
      IRTable[] t;
```

```
        fillTable(t); // Fills entries such that t[i]->handle =
table[OPCODE];
        i=0;
        while (t[i])
        {
                t[i]->handle();
        }
}

void handle_jump() { …}
void handle_print() { …}
void handle_add() { …}
void handle_halt() { …}
```

18. [6 points] You are given an empty 16K 2-way set-associative LRU-replacement cache with 32 byte blocks on a machine with 4 byte words and 32-bit addresses. Describe in mathematical terms a memory read address sequence which yields the following Hit/Miss patterns. If such a sequence is impossible, state why. Sample sequences:

address(N) = N mod 2^{32} (= 0, 1, 2, 3, 4...)
address = (7, 12, 14)

(a) Miss, Hit, Hit, Miss
(b) Miss, (Hit)*
(c) (Hit)*
(d) (Miss)*
(e) (Miss, Hit)*

The number of sets in this cache is = 16 * (2^{10})/2*2^5
Each set has two cache blocks, each holding 32 bytes.

(a) Miss, Hit, Hit, Miss : 0,0,0,32
(b) Miss, (Hit)* : 0, 0*
(c) (Hit)* : Such a sequence is not possible since we begin with an empty cache and there has to be at least one compulsory miss
(d) (Miss)* : 0,32, 64, 64+32, …, 32*N, 0, 32, 64, … where N = 2^{32}/16*(2^{10}) = 2^{18}
(e) (Miss, Hit)* : 0, 0, 32, 32, 64, 64, 64 + 32, 64+32, …, 32*N, 32*N, 0, 0, 32, 32, 64, 64 … where N = 2^{32}/16*(2^{10}) = 2^{18}

19. [3 points] Assume an instruction cache miss rate for gcc of 2% and a data cache miss rate of 4%. If a machine has a CPI of 2 without any memory stalls and the miss penalty is 40 cycles for all misses, determine how much faster a machine would run with a perfect cache that never missed. Assume 36% of instructions are loads/stores.

Let the instruction count be I
Instruction cache Stalls = I * 0.02 * 40 = 0.8I
Data cache stalls = I * 0.36 * 0.04 * 40 = 0.56I
Total memory stalls = 1.36 * I

CPI with perfect memory = 2.0

CPI with memory stalls = 3.36

Perfect memory performance is better by 3.36/2 or 1.66

20. [12 points] Caching. "One of the keys to happiness is a bad memory." –Rita Mae Brown

Consider the following piece of code:

```
int x = 0, y = 0; // The compiler puts x in r1 and y in r2.
int i; // The compiler put i in r3.
int A[4096]; // A is in memory at address 0x10000
...
for (i=0;i<1024;i++) {
x += A[i];
}
for (i=0;i<1024;i++) {
y += A[i+2048];
}
```

(a) Assume that the system has a 8192-byte, direct-mapped data cache with 16-byte blocks. Assuming that the cache starts out empty, what is the series of data cache hits and misses for this snippet of code. Assume that ints are 32-bits.

Each block has 16-bytes or 4 ints. Since the cache is a direct mapped one,
for the first loop, one in 4 elements causes a miss. Sequence: Miss, Hit, Hit, Hit, Miss, Hit, Hit, Hit, Miss, …

Number of misses = 1024/4 = 256 , Number of Hits = 768

For the second loop: 2048*4 = 8192 bytes which is the capacity of the direct mapped cache.Therefore A[i+2048] is again mapped starting from 0 onwards. So the sequence is same above: Miss, Hit, Hit, Hit, Miss, Hit, Hit, Hit, Miss, …

Number of misses = 1024/4 = 256 , Number of Hits = 768

Total Number of misses = 512 , Number of hits = 1536

(b) Assume that an iteration of a loop in which the load hits takes 10 cycles but that an iteration of a loop in which the load misses takes 100 cycles. What is the execution time of this snippet with the aforementioned cache?

Discounting such factors like branch prediction etc,

Total Execution Time = 10*1536 + 100*512
= 66560 cycles

(c) Repeat part A except assume that the cache is 2-way set associative with an LRU replacement policy and 16-byte sets (8-byte blocks).

The number of sets in this case will be 8192/16 = 1024.
Each block in a set holds 2 ints.

Name:_____

Since 1024 accesses are consecutive in the first loop, there is one miss per 2 elements.

Number of misses = 1024/2 = 512 , Number of Hits = 512

The second loop accesses elements that in the same set as the first loop and hence results in the same miss behavior.

Number of misses = 1024/2 = 512 , Number of Hits = 512

Total Number of misses = 1024 , Number of hits = 1024

(d) Repeat part B using the cache described in part C. Is the direct-mapped or the set-associative cache better?

Total Execution Time = 10*1024 + 100*1024
= 112640 cycles

The set-associative cache is better