CMPE 110 Computer Architecture, Fall 2014
Homework #5

Computer Engineering
UC Santa Cruz

November 14, 2014

Name: _____

Email: _____

Submission Guidelines:

- This homework is due on Thursday 11/20/14.
- Bring the homework to class before 8pm
  - Anything later is a late submission
- **Please write your name and your UCSC email address**
- **The homework should be "readable" without too much effort**
  - If your handwriting is like mine, type it or risk not being graded
- Provide details on how to reach a solution. An answer without explanation has no credit. Clearly state all assumptions.
- Points: 44 = 14 + 10 + 20

Question	Part A	Part B	Part C	Part D	Part E	Part F	Total
1						-	
2				-	-	-	
3							
Total							

Question 1. Basic Cache (14 points)

Consider a 1-MByte cache with 16-word cachelines (a cacheline is also known as a cache block, each word is 4-Bytes). This cache uses write-back scheme, and the address is 64-bits wide.

Question 1.A Direct-Mapped, Cache Fields (2 points)

Assume the cache is direct-mapped. Fill in the table below to specify the size of each address field.

Field	**Size (bits)**
Byte Offset	
Cacheline Offset	
Cacheline Index	
Tag	

Question 1.B 4-Way Set-Associative, Cache Fields (1 point)

Assume the cache is 4-way set-associative. Fill in the table below to specify the size of each address field.

Field	**Size (bits)**
Byte Offset	
Cacheline Offset	
Cacheline Index	
Tag	

Question 1.C Direct-Mapped, Cache Transactions (6 points)

Assume the cache is direct-mapped. Fill in the table on the next page to identify the content of the cache after each of the following memory accesses. Assume the cache is initially empty (also called a "cold cache"). Specify if an entry causes another line to be replaced from the cache, and if an entry has to write its data back to memory. For the data column, specify the data in the block by referring to its address like M[address]. Write accesses will modify the data, so let's indicate the data after a write access with D[address].

Address	Request	Cachline Index	Valid	Modified	Tag	Data	Cause Replaced?	Write-back to Memory?
0x128	read					M[0x120]		
0xF40	write					D[0xF40]		
0xC00024	read							
0x014	write							
0x1000F44	read							

Question 1.D 4-Way Set-Associative, Cache Transactions (3 points)

Assume the cache is 4-way set-associative. Fill the table below to identify the content of the cache after each of the following memory accesses. Assume the cache is empty in the beginning (also known as cold cache). Specify if an entry causes another line to be replaced from the cache, and if an entry has to write its data back to memory. For the data column, specify the data in the block by referring to its address like M[address]. Write accesses will modify the data, so lets indicate the data after a write access with D[address].

Address	Request	Cachline Index	Valid	Modified	Tag	Data	Cause Replaced?	Write-back to Memory?
0x128	read					M[0x120]		
0xF40	write					D[0xF40]		
0xC00024	read							
0x014	write							
0x1000F44	read							

Question 1.E Overhead (2 points)

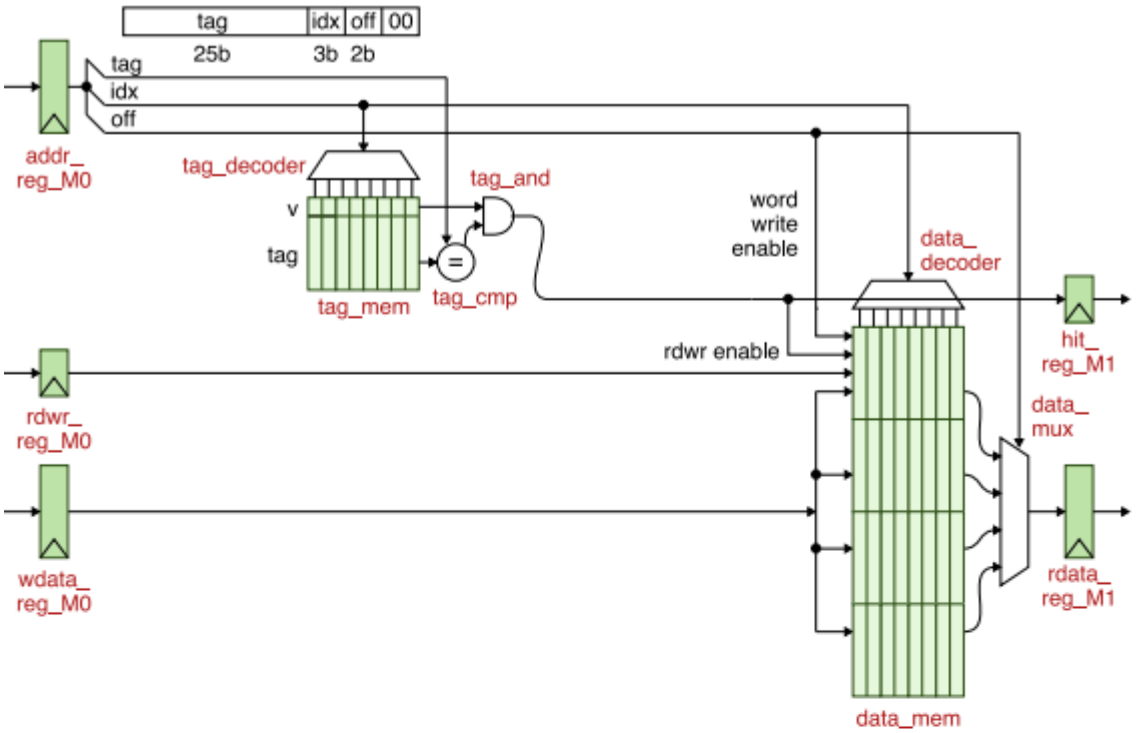
What is the overhead and actual size of the direct-mapped cache? What is the overhead and actual size of the 4-way set-associative cache? Does the structure change the overhead in terms of number of memory bits?

Question 2. Impact of Cache Access Time (10 points)

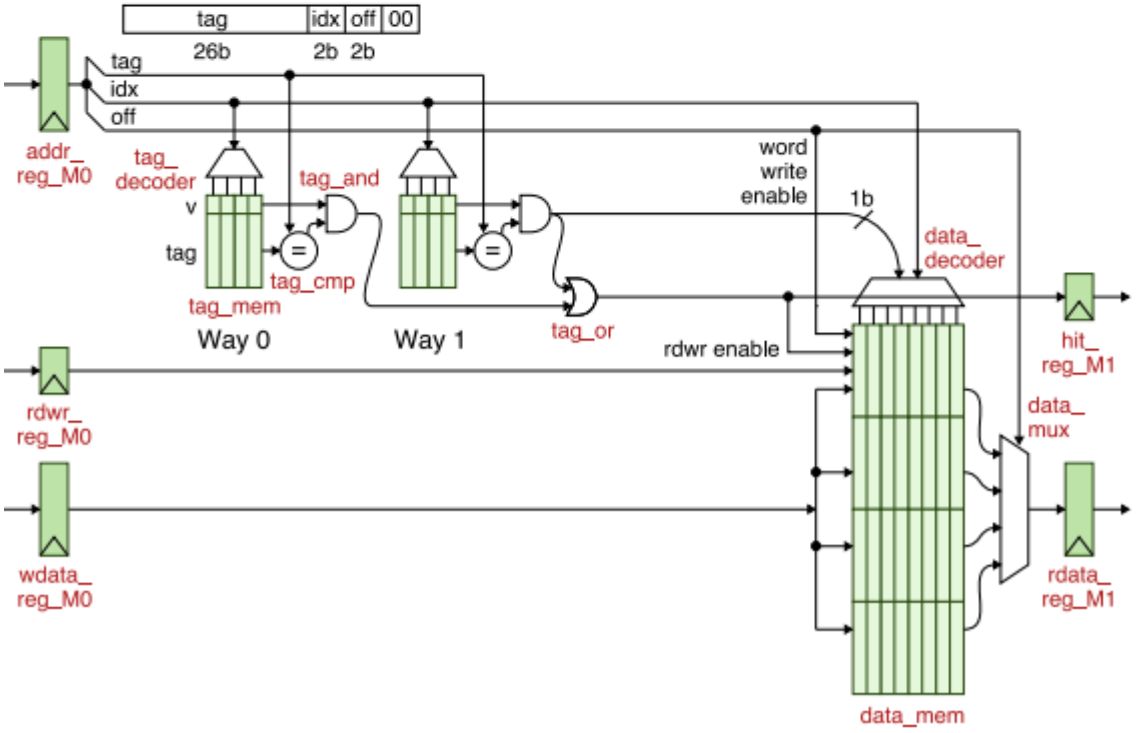
In this problem, we will be comparing various microarchitectures for a simple data cache. For all parts, the memory requests use a 32-bit address, although you should assume that all addresses are word aligned. Since words are four bytes, this means the bottom two bits of the address will always be zero. All caches contain exactly eight cache lines, and each cache line contains four words (i.e., each cache line is 16 bytes long). Thus the total cache capacity is 8×16 B $= 128$ B.

This problem will require you to identify the critical path of a specific microarchitecture. The table below lists simplified delay equations for the cache hardware components. These delay equations are parameterized by the size of each component. Delay is measured in normalized gate delays, where 1τ is the delay of a single inverter driving four identical inverters. To simplify things, assume that the delay of a component is always the same regardless of the order in which different inputs arrive at a component. More specifically, the delay of a write access is the same regardless of whether the address arrives before the write data or vice versa. Also assume that we are using combinational memories (i.e., the address is set and the data is returned on the same cycle). Note that the $\lceil x \rceil$ notation denotes the ceiling operator, i.e., the value x rounded up to the next largest integer.

Components	**Delay (τ)**	**Comment**
register read	1	delay from clock edge to data out
register write	1	setup time constraint
n-input AND gate	n	simple logic gate
n-input OR gate	n	simple logic gate
n-to-m decoder	$2 + 2n$	2 for fixed overhead, $2n$ for logic
n-bit comparator	$3 + 2\lceil \log_2 n \rceil$	3 for initial XOR gate, $2\lceil \log_2 n \rceil$ for OR tree
n-input mux	$2 + 3\lceil \log_2 n \rceil$	2 for fixed overhead, $3\lceil \log_2 n \rceil$ for tree-based muxing logic
$n \times m$ decoder	$10 + \lceil (n + m)/32 \rceil$	n rows and m bits per row, 10 for fixed overhead and bitcell access, $\lceil (n + m)/32 \rceil$ to drive word and bit lines



(a) Direct-Mapped Cache



(b) Two-Way Set-Associative Cache

Question 2.A Sequential Tag Check then Memory Access (2 points)

The diagram on the previous page illustrates two cache microarchitectures that serialize the tag check before data access. This means that for both reads and writes, the cache completely finishes the tag check and accesses the data memory only on a cache hit. figure (a) is for a directed-mapped cache, while figure (b) is for a two-way set-associative cache.

We now want to determine the critical path and cycle time in units of τ for each cache microarchitecture. As an example, the table below shows the critical path and cycle time for the directed-mapped cache from (a). Note that because we are serializing the tag check before data access and because the delay equations are the same for both read and write accesses, the critical path is the same regardless of whether we are doing a read or a write. This is a simplification, but it will do for the purposes of this part. Note that the tag is 25 bits, but each row of the tag memory requires 26 bits since it must also include a valid bit.

Create a table similar to the one in the example which identifies the critical path and cycle time in units of τ for the two-way set-associative cache in (b). Compare the cycle times of the two cache microarchitectures. What is the primary reason one microarchitecture is slower than the other microarchitecture?

Component	**Delay Equation**	**Delay (τ)**
`addr_reg_M0`	1	1
`tag_decoder`	$2 + 2 \cdot 3$	8
`tag_mem`	$10 + \lceil (8 + 26)/32 \rceil$	12
`tag_cmp`	$3 + 2\lceil \log_2 25 \rceil$	13
`tag_and`	2	2
`data_mem`	$10 + \lceil (8 + 128)/32 \rceil$	15
`data_mux`	$2 + 3\lceil \log_2 4 \rceil$	8
`rdata_reg_M1`	1	1
Total		**60**

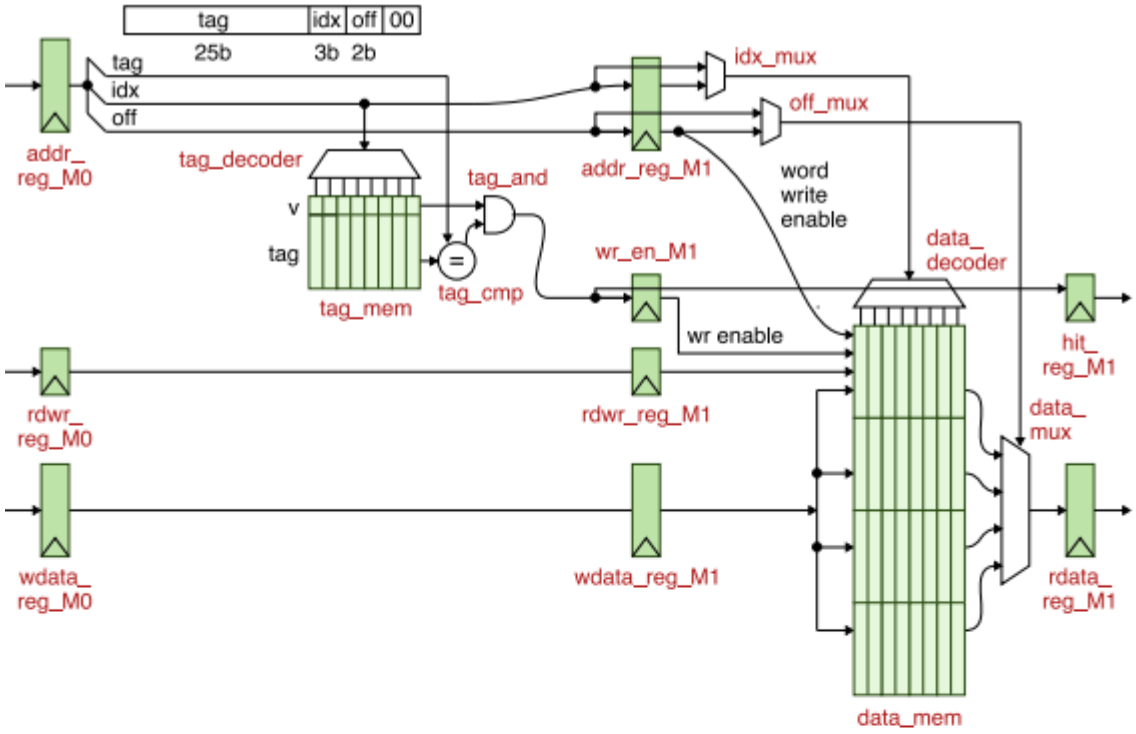
Question 2.B Parallel Read Hit Path (4 points)

The diagram on the next page illustrates two cache microarchitectures with parallel read hit paths and pipelined write hit paths. This means that for a single read request, the tag check is done in parallel with the data memory read access, while for for a single write request the tag check is done in stage M0 and the data memory write access is done in stage M1. Figure (a) is for a directed-mapped cache, while figure (b) is for a two-way set-associative cache.

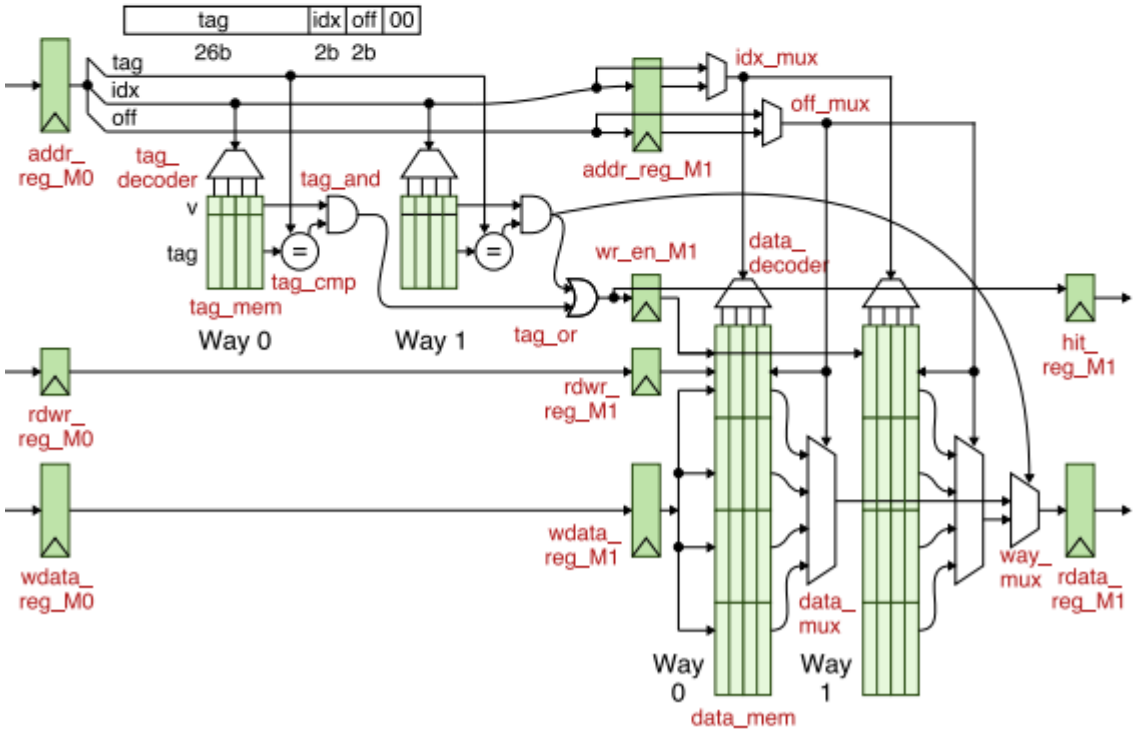
For this part we will *focus just on the parallel read hit path* for both the direct-mapped and set-associative caches. Create two tables similar to the one from the previous part which identifies the critical path and cycle time in units of τ for just the parallel read hit paths. Note that since the tag check and the data memory read access are done in parallel, you will need to examine both of these paths to determine which one is in fact the critical path. Compare the cycle times of the two cache microarchitectures. What is the primary reason one microarchitecture is slower than the other microarchitecture?

Question 2.C Pipelined Write Hit Path (4 points)

For this part we will focus just on the pipelined write hit path for both the direct-mapped and set-associative caches shown on the next page. Create two tables similar to the one in the previous parts which identifies the critical path and cycle time in units of τ for just the pipelined write hit path. Note that since the tag check and the data memory write access happen in two different stages, you will need to examine both of these paths to determine which one is in fact the critical path. Compare the cycle times of the two cache microarchitectures. What is the primary reason one microarchitecture is slower than the other microarchitecture?



(a) Direct-Mapped Cache



(b) Two-Way Set-Associative Cache

Question 3. Two-Cycle Instruction Cache (20 points)

In this problem, we will be examining the performance of the instruction cache on the MIPS assembly program shown on the next page. The first column shows the instruction address for each instruction. Note that these addresses are byte addresses. The value of r1 is initially 64, meaning that there are 64 iterations in the loop. In this problem, we will be considering the execution of this loop with a direct-mapped instruction cache microarchitecture with eight cache lines, and each cacheline is 16B. This means each cache line can hold four instructions and the bottom four bits of an instruction address are the block offset. Hint: The first instruction in the code segment (i.e., `addiu r1, r1, -1`), is in the middle of a cache line.

For this problem, the instruction cache hit time is two cycles, but it is fully pipelined. Tag check occurs in the first cycle, and if it is a hit, the instruction is read in the second cycle. Essentially, this creates a six-stage pipelined processor with the following stages: instruction cache tag check (F0), instruction cache data access (F1), decode (D), execute (X), memory (M), and write-back (W). This also implies the data cache hit time is one cycle.

Assume that jumps are resolved in the decode stage and that branches are resolved in the execute stage. Assume the miss penalty is three cycles so on a cache miss the pipeline will stay in F0 for a total of three cycles, go into F1 for one cycle, and then continue as normal. You should assume that in every other way, the processor pipeline follows the classic fully-bypassed five-stage pipeline. Assume that the processor does not include a branch delay slot. Assume the processor speculatively predicts all jumps and branches are not taken.

Question 3.A Control Hazards (3 points)

Draw a pipeline diagram illustrating the first iteration of the loop assuming there are no instruction cache misses. Remember that there are two fetch stages (F0 and F1). Show stalls by simply repeating the pipeline stage character (e.g., D) for multiple consecutive cycles. Use a dash (-) to indicate pipeline bubbles caused by killing instructions. Draw two arrows indicating the control dependencies for the jump and branch instructions. The arrow should start in the stage where the jump/branch is resolved and point to the first fetch stage of the target instruction. You should show all instructions in the first iteration of the loop and the first instruction of the second iteration that you can properly draw the control dependency for the backwards branch.

Address	Instruction	Q4.B Iteration 1 ICache Miss Type	Q4.C Iteration 2 ICache Miss Type
	`loop:`		
0x108	`addiu r1, r1, -1`		
0x10c	`j foo`		
0x110	`addiu r2, r2, 1`		
0x114	`addiu r3, r3, 1`		
0x118	`addiu r4, r4, 1`		
0x11c	`addiu r5, r5, 1`		
	`...`		
	`foo:`		
0x218	`bgtz r1, loop`		
0x21c	`addiu r6, r6, 1`		
0x220	`addiu r7, r7, 1`		
0x224	`addiu r8, r8, 1`		
0x228	`addiu r9, r9, 1`		

Question 3.B First Iteration of the Loop (4 points)

Fill in the table above. In the appropriate column, write *compulsory*, *conflict*, or *capacity* next to each instruction which misses in the instruction cache to indicate the type of instruction cache misses that occur in the first iteration of the loop. **Assume that the instruction cache is initially completely empty.** Now draw a pipeline diagram illustrating the first iteration of the loop including instruction cache misses. Assume that a given instruction can only stall instructions that are after it in program order, and can never stall instructions that are before it in program order. Clearly indicate the number of cycles it takes to execute the first iteration and also indicate the instruction cache miss rate. The miss rate includes instructions that are fetched but then later squashed. As in the previous part, draw arrows indicating the control dependencies for the jump and branch instructions.

Question 3.C Second Iteration of the Loop (4 points)

Continue to fill in the table above. Write *compulsory*, *conflict*, or *capacity* next to each instruction which misses in the instruction cache to indicate the type of instruction caches misses that occur in the second iteration of the loop. Now draw a pipeline diagram illustrating the second iteration of the loop. Clearly indicate the number of cycles it takes to execute the second iteration and the instruction cache miss rate. The miss rate includes instructions that are fetched but then later squashed. As in the previous part, draw arrows indicating the control dependencies for the jump and branch instructions.

Question 3.D Average Memory Access Latency (3 points)

Calculate the average instruction cache memory access latency in cycles for 64 iterations of the loop. You must show your work, especially the various components of the average memory access latency. Remark on which kind of miss is dominating the average memory access latency.

Question 3.E Processor Performance (3 points)

Calculate the CPI for this processor executing all 64 iteration of the loop. You must show your work. Note that the "I" in CPI stands for instruction and that we do not include instructions that are fetched but then later squashed in this count. Similarly, the CPI due to executing useful work, should not include instructions that are fetched but then later squashed.

Question 3.F Set-Associativity (3 points)

Qualitatively, predict how the cache performance would change if we replace the eight-entry, direct-mapped cache with an eight-entry, two-way, set-associative cache. Both caches have a two-cycle hit latency. Assume the set-associative cache address interleaves the sets across the ways using the least significant bits right after the block offset. What kind of misses would be present with this kind of cache microarchitecture?