CSE 141: Introduction to Computer Architecture Spring 2024

Homework 3

Instructor: Jishen Zhao **Due on:** May 24, 2024 *(50 points)*

Name: ______________________
PID: ______________________
Email: ______________________

Instructions

- The homework must be submitted to Gradescope by 11:59pm. Anything later is a late submission
- Handwritten or typed responses are accepted.
- All responses must be neat and legible. Illegible answers will result in zero points.
- Provide details on how to reach a solution. An answer without explanation gets no credit. Clearly state all assumptions.

1. **Instruction Cache. (17 points)**

In this problem, we will be exploring adding an instruction cache to a standard 5-stage pipe-lined processor. We will be using the MIPS assembly program shown below. The first column shows the instruction address for each instruction. Note that these addresses are byte addresses. The value of r1 is initially 32, meaning that there are 32 iterations in the loop. In this problem, we will be considering the execution of this loop with a direct-mapped instruction cache micro-architecture with eight 16-Byte cache lines. This means each cache line can hold four instructions and the bottom four bits of an instruction address are the block offset.

Address	Instruction	Iteration 1	Iteration 2	Iteration 3
	loop:			
0x208	addiu r1, r1, -1	*compulsory*		
0x20c	addiu r2, r2, 1			
0x210	addiu r3, r3, 1	*compulsory*		
0x220	j foo	*compulsory*	*conflict*	*conflict*
	...			
	foo:			
0x320	addiu r6, r6, 1	*conflict*	*conflict*	*conflict*
0x324	bne r1, r0, loop			

Question 1.A Categorizing Cache Misses (8 points)
Fill out the table above. In the appropriate columns, write *compulsory*, *conflict*, or *capacity* next to each instruction that misses in the instruction cache to indicate the type of instruction cache misses that occur in the first, second and third iterations of the loop. Assume that the instruction cache is initially completely empty.

- *Compulsory misses* occur when a memory access maps to a currently invalid (or empty) cache line or cache set.
- *Conflict misses* occur when multiple memory addresses map to the same cache line or cache set.
- *Capacity misses* occur when the cache is full and can no longer handle further memory access requests or contain the working set of data needed for program execution.

Solution:
See the table above. The first instruction fetch causes a compulsory miss, but then we hit in that cache line when fetching the second addiu instruction. The third addiu instruction will cause another compulsory miss as it maps to a new cache line during the first iteration. During the first iteration, the jump instruction causes a compulsory miss. The target of the jump instruction maps to the same set as the jump instruction maps to, so this is a conflict miss. In the remaining iterations, the cache lines at addresses 0x220 and 0x320 will continue to conflict in the cache every iteration while other instruction fetches will always hit the correct cache line
0x208: bit: 1000001000, tag: 100, index: 000, byte offset: 1000
0x20c: bit: 1000001100, tag: 100, index: 000, byte offset: 1100
0x210: bit: 1000010000, tag: 100, index: 001, byte offset: 0000
0x220: bit: 1000100000, tag: 100, index: 010, byte offset: 0000
0x320: bit: 1100100000, tag: 110, index: 010, byte offset: 0000
0x324: bit: 1100100100, tag: 110, index: 010, byte offset: 0100

Question 1.B Average Memory Access Latency (5 points)
Calculate the instruction cache miss rate for 32 iterations of the loop. Calculate the average instruction cache memory access latency in cycles for 32 iterations of the loop. Assume the hit time is one cycle and that the miss penalty is 4 cycles. 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.

Solution:
Each iteration performs 6 instruction cache accesses. There are 4 misses on the first iteration, and two misses on each remaining iteration for a total of $4 + 2 \times 31 = 66$ misses. So the total miss rate is $66/192 = 0.34375$. Avg Mem Access Latency = Hit Time + (Miss Rate Miss Penalty) = $1 + (0.34375 \times 4) = 2.375$ cycles Clearly the fixed overhead of the extra compulsory cache miss in the first iteration is largely irrelevant compared to the conflict misses in the remaining iterations of the loop.

Question 1.C Set-Associativity (4 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 one-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 micro-architecture?

Solution:
Since the majority of the instruction cache misses are conflict misses, it is possible that increasing the associativity will reduce the number of instruction cache misses. More specifically, two lines are conflicting in the same set of the direct-mapped cache, so a two-way set-associative cache will be able to keep both lines present in the cache at the same time. Of course, we need to also ensure that the third cache line doesn't map to the same set, which in this case a quick look at the addresses verifies that it does not. So a two-way set-associative instruction cache should have a miss rate close to zero on this loop. The only misses would be three compulsory misses and one conflict miss on the first iteration of the loop.

2. **Cache Mapping and Access (21 points)**

Consider a 1024-KByte cache with 32-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 32 bits wide. Clearly show your calculations to receive full marks.

Question 2.A Direct-Mapped, Cache Fields (3 points)
Assume the cache is direct-mapped. Fill in the table below to specify the size of each address field.

Field	Size (bits)
Cacheline Offset	7
Cacheline Index	13
Tag	12

Solution:
Offset = log2(32 × 4) = 7 bits. Index = log2(1024KB/ (32*4)) = 13 bits. Tag = 32 - 7 - 13 = 12 bits.

Question 2.B Fully-Associative, Cache Fields (3 points)
Assume the cache is fully-associative. Fill in the table below to specify the size of each address field.

Field	Size (bits)
Cacheline Offset	7
Cacheline Index	0
Tag	25

Solution:
Offset = log2(32 × 4) = 7 bits. Same as in part A. Fully-associative caches do not use the index field. Tag = 32 - 7 = 25 bits.

Question 2.C 8-Way Set-Associative, Cache Fields (3 point)
Assume the cache is 8-way set-associative. Fill in the table below to specify the size of each address field.

Field	Size (bits)
Cacheline Offset	7
Cacheline Index	10
Tag	15

Solution:
Offset = log2(32 × 4) = 7 bits. Index = log2(1024KB/ (32*4*8)) = 10 bits. Tag = 32 - 7 - 10 = 15 bits.

Question 2.D Tag Overhead (12 points)
(i) What is the tag overhead and total size of the direct-mapped cache?
Considering the following conditions:
a. Tag overhead includes the valid bit and tag bits
b. Tag overhead includes the valid bit and the dirty bit as well as tag bits

Solution:
Direct-Mapped Cacheline size: number of words (32) × size of word (4) × 8 bits/byte = 1024 bits Total number of cache lines: 1024KB / (32*4) = 8192
Overhead:
1. If we include tag bits (12) + valid bit (1)
Actual Size = (no. of cachelines × size of cacheline) + (no. of cachelines × overhead fields size) = (8192 × 1024) + (8192 × 13)
Final Answer in terms of bits: 8388608 bits + 106496 bits = 8495104 bits
Final Answer in terms of bytes: 1048576 bytes + 13312 bytes = 1061888 bytes
Final Answer in terms of kilobytes: 1024 KB + 13 KB = 1037KB
Tag overhead = 13KB / 1024KB * 100% = 1.27%

2. If we include tag bits (12) + valid bit (1) + dirty bit (1)
Actual Size = (no. of cachelines × size of cacheline) + (no. of cachelines × overhead fields size) = (8192 × 1024) + (8192 × 14)
Final Answer in terms of bits: 8388608 bits + 114688 bits = 8503296 bits
Final Answer in terms of bytes: 1048576 bytes + 14336 bytes = 1062912 bytes
Final Answer in terms of kilobytes: 1024 KB + 14 KB = 1038 KB
Tag overhead = 14KB / 1024KB * 100% = 1.37%

(ii) What is the tag overhead and total size of the 8-way set-associative cache?
Considering the following conditions:
a. Tag overhead includes the valid bit and tag bits
b. Tag overhead includes the valid bit, the dirty bit, and tag bits

Solution:
8-Way Set-Associative Cache Cache set size: number of words (32) $\times$ size of word (4) $\times$ number of cacheline per set (8) $\times$ 8 bits/byte = 8192 bits
Total number of cache sets: 1024KB/ (32*4*8) = 1024
However, the total number of cache lines remains the same: 1024KB/(32*4) = 8192, and we'll use this number again in the calculation of overhead

Overhead:
1. If we include tag bits (15) + valid bit (1)
Actual Size = (no. of cache sets $\times$ size of cache set) + (no. of cachelines $\times$ overhead fields size) = (1024 $\times$ 8192) + (8192 $\times$ 16)
Final Answer in terms of bits: 8388608 bits + 131072 bits = 8519680 bits
Final Answer in terms of bytes: 1048576 bytes + 16384 bytes = 1064960 bytes
Final Answer in terms of kilobytes: 1024 KB + 16.0 KB = 1040 KB
Tag overhead = 16KB / 1024KB * 100% = 1.56%

2. If we include tag bits (15) + valid bit (1) + dirty bit (1)
Actual Size = (no. of cache sets $\times$ size of cache set) + (no. of cachelines $\times$ overhead fields size) = (1024 $\times$ 8192) + (8192 $\times$ 17)
Final Answer in terms of bits: 8388608 bits + 139264 bits = 8527872 bits
Final Answer in terms of bytes: 1048576 bytes + 17408 bytes = 1065984 bytes
Final Answer in terms of kilobytes: 1024 KB + 17 KB = 1041 KB
Tag overhead = 17KB / 1024KB * 100% = 1.66%

3. **Average Memory Access Time (12 points)**
Considered two pipelined machines A and B, are described in the table below.

Property	Computer A	Computer B
ISA	MIPS	x86
Clock Frequency	3 GHz	4 GHz
Base CPI	2 cycles/instruction	3 cycles/instruction
L1 Instruction Cache Hit Time	1 cycle	1 cycle
L1 Instruction Cache Miss Rate	4%	4%
L1 Data Cache Hit Time	1 cycle	2 cycles
L1 Data Cache Miss Rate	8%	5%
Main Memory Access Time	200 cycles	250 cycles

Question 3.A Ideal System (6 points)
Assume a perfect memory system (100% of memory accesses hit in the L1 caches) and perfect branch prediction. Assume that MIPS programs execute 1.3x as many instructions as x86 programs.
Which machine has the faster execution time, and what is the speedup?

Solution:
The hit time of the cache is already included in the base CPI calculation for question 3.
In an ideal system with no cache misses, systems behave according to base CPI.
B Execution Time = IC $\times$ CPI $\times$ clock cycle time = ICB $\times$ 3 $\times$ 250 ps
A Execution Time = IC $\times$ CPI $\times$ clock cycle time = ICA $\times$ 2 $\times$ 333 ps = (1.3 $\times$ ICB) $\times$ 2 $\times$ 333 ps
Speedup = A Execution Time (slow) / B Execution Time (fast)
Speedup = (1.3*2*333) / (1*3*250) = 1.1544
Computer B is faster by 15.44%

Question 3.B No L2 Cache (6 points)
Now assume each machine has only L1 caches (split iL1 cache and dL1 cache), no L2 cache, perfect branch prediction, and that 25% of the instructions are loads/stores.
Which machine has the faster execution time, and what is the speedup?

Solution:
CPI = Base CPI + stall IL1 + stall DL1 = Base CPI + (missrate IL1 $\times$ misspenalty IL1)+(25% $\times$ (missrateDL1 $\times$ misspenaltyDL1)
CPI A = $2 + 0.04 \times 200 + 0.25 \times 0.08 \times 200 = 14$ cycles/instruction
CPI B = $3 + 0.04 \times 250 + 0.25 \times 0.05 \times 250 = 16.125$ cycles/instruction
B Execution Time = IC $\times$ CPI B $\times$ clock cycle time = ICB $\times$ 16.125 $\times$ 250 ps
A Execution Time = IC $\times$ CPI A $\times$ clock cycle time = ICA $\times$ 14 $\times$ 333 ps = (1.3 $\times$ ICB) $\times$ 14 $\times$ 333 ps
Speedup = A Execution Time (slow) / B Execution Time (fast)
Speedup = (1.3*14*333) / (1*16.125*250) = 1.50340465
Computer B is faster by 50.3%