CS 250 Spring 2017 Homework 09
Due 11:58pm Wednesday, April 05, 2017
Submit your typewritten file in PDF format to Blackboard.

1. If a cache is big enough for the 90/10 Rule to apply, and the cache is 10 times faster than main memory, what does Amdahl's Law predict for the overall speedup from using this cache?

   $speedup = \frac{1}{(1-p)+{}^{p}/_{s}}$ p is the proportion of execution time sped up

   S is the speedup of the part of the task that benefits

   With p = 0.9 and s = 10….
   speedup = 5.26 by using the cache.

2. How many direct-mapped cache organizations are possible given 32-bit byte addressing for programmers, word-addressed physical memory, 4-byte words, and a 17-bit tag?
   Given the following information, we can conclude that the address will be 32 bits. With a 17-bit tag this leaves 15 bits for the cache line index and offset. Since each word is 4 bytes this means the offset must be 5 bits, thus leaving 10 bits for the cache line index. **In conclusion this means that there are 2^10, or 1024, direct-mapped caches.**

3. The i-cache (instruction cache) for a 32-bit RISC processor is direct-mapped with an offset field of 4 bits. Assume the cache is cold. Soon, the processor will fetch its first instruction at address 0x00400000.
   a. What is the best assumption about the size(s) of instructions for this processor? Why is it almost certain that the processor is designed this way?
      The best assumption about the size of the instruction is 32 (NOT 64) because the RISC processor works at higher performance with a simplified instruction set. It is almost certain that the processor is designed this way because it improves performance when paired with other hardware (like a processor should), while also using fewer microprocessor cyclers per instruction.

   b. How many 32-bit instructions can a block in this i-cache (instruction cache) hold?
      2^10 = 1024 instructions

   c. Assume that the program loaded into memory starting at address 0x00400000 is pure straight line code. This means that this program is one, giant basic block of code, who knows how long, that fits within the available memory. Describe the hit and miss behavior of this i-cache (instruction cache) as the processor fetches the first four instructions of the program. State the type(s) of miss(es) that occur. Carefully consider how the address of the first instruction affects the number of hits and misses.

      1. 0x00400000 compulsory (cold) miss
      2. 0x00400010 hit
      3. 0x00400020 compulsory (cold) miss
      4. 0x00400030 hit

d. Now the program is loaded into memory starting at address 0x00400008. What is the hit/miss behavior and miss type(s) for the first four instructions? What happens afterwards?

5. 0x00400008 compulsory (cold) miss
6. 0x00400018 compulsory (cold) miss
7. 0x00400028 compulsory (cold) miss
8. 0x00400038 compulsory (cold) miss

e. Expressed as a percentage, what is the overall hit rate of the straight line program? Remember, the program is many instructions long.
The overall hit rate of the straight-line program is around 50%.

f. Now the program contains more than just straight line code. It has been re-written to have straight line code plus the occasional if-then-else statement. The program looks like this around an if-then-else statement (based Figure 9.2 in our textbook).

High level program	Assembly code			Basic Block
statement ;		instrs. for statement	; straight line (SL) instructions	1
if (cond.) {		instr. for condition	; 1 instruction evaluates condition	1
		branch to else	; EXIT the SL basic block	1
then_part		instrs. for then_part	; section of SL, a basic block	2
		…	; remaining then_part SL instrs.	2
} else {		branch to next_stmt	; EXIT the SL "then" basic block	2
else_part	else:	instrs. for else_part	; TARGET, else_part basic block	3
}		…	; remaining else_part SL instrs.	3
next_stmt ;	done:	instrs. for next_stmt	; TARGET, fall-through from else_part continues SL	3

Assume that the address of any given assembly instruction (machine instruction) in the assembly code is equally likely to correspond to any one of the 4 possible word offsets (w_offset) within an i-cache block. This means that the execution trace of the program can exit straight line code when executing any instruction word in a cache block.

Just for cache blocks containing a branch instruction, what is the average hit rate expressed as a percentage?

25% hit rate

For simplicity in analyzing the situation for this question, assume (1) that any cache block contains at most one branch instruction (just one chance for the program

execution trace to exit the block early) and (2) if a cache block contains a branch instruction, then it does not also contain a target instruction (a way back for the program execution trace to get back into the block after exiting because of the branch).

g. In part (f), the inclusion of if-then-else in the original program, which was only straight line code, worsened the hit rate for the i-cache. Now imagine adding loops to the program of part (f). Will the hit rate further worsen, stay the same, or improve? Using the program instruction execution trace concept, explain the reasoning behind your prediction of the effect of loops on hit rate.
I believe the loops will further worsen the hit rate. This is because I predict the increased size of the block of code alongside many other factors of this now complex loop-statement all add up to decrease and thus worsen the hit rate. In other words, new types of misses may arise now and thus more misses in general.

h. The i-cache of this question has been designed with 4-word blocks. Think of i-cache operation in the event of a cache miss as an opportunity to perform instruction execution trace prediction. As compared to an i-cache design alternative where the block holds only one word, a 4-word-block design embodies its designer's vote for which type of execution trace prediction: no prediction, predict default_next, predict not default_next? Pick your answer and then explain why it is the correct choice.
'predict default_next'. This is the obvious choice because as the designer we would absolutely want to predict the default next so we can reduce misses and increase the hit rate and thus improve overall memory access time.

i. Cache misses can be categorized into three groups: compulsory miss, capacity miss, and conflict miss. In question part (g) program loops were introduced. Which miss type(s) can occur in the program with loops that are not possible for the program with only straight line code and if-then-else constructs?
Capacity miss and conflict miss are not possible for programs with straight line code and if-then-else constructs.

4. A faster replacement algorithm for an associate cache improves which term or factor of the average memory access time equation?
A faster replacement policy for an associate cache will also improve the hit ratio, which in turn makes the 'miss rate' lower. In conclusion, this makes the overall average memory access time faster.