Computer Architecture, Problem Set #1A

Note: We have procured permission to post these problems from Computer Architecture: A Quantitative Approach from the publisher, and will be using the original problems as the supplement.

Problem #3 (20 Points): Page C-82 in H&P5, Problem C.1 a,b,c,d,e,f,g

C.1 [15/15/15/15/25/10/15] <A.2> Use the following code fragment:

```
Loop:   LD      R1,0(R2)    ;load R1 from address 0+R2
        DADDI   R1,R1,#1    ;R1=R1+1
        SD      R1,0,(R2)   ;store R1 at address 0+R2
        DADDI   R2,R2,#4    ;R2=R2+4
        DSUB    R4,R3,R2    ;R4=R3-R2
        BNEZ    R4,Loop     ;branch to Loop if R4!=0
```

Assume that the initial value of R3 is R2 + 396.

a. [15] <C.2> Data hazards are caused by data dependences in the code. Whether a dependency causes a hazard depends on the machine implementation (i.e., number of pipeline stages). List all of the data dependences in the code above. Record the register, source instruction, and destination instruction; for example, there is a data dependency for register R1 from the LD to the DADDI.

b. [15] <C.2> Show the timing of this instruction sequence for the 5-stage RISC pipeline without any forwarding or bypassing hardware but assuming that a register read and a write in the same clock cycle "forwards" through the register file, as shown in Figure C.6. Use a pipeline timing chart like that in Figure C.5. Assume that the branch is handled by flushing the pipeline. If all memory references take 1 cycle, how many cycles does this loop take to execute?

c. [15] <C.2> Show the timing of this instruction sequence for the 5-stage RISC pipeline with full forwarding and bypassing hardware. Use a pipeline timing chart like that shown in Figure C.5. Assume that the branch is handled by predicting it as not taken. If all memory references take 1 cycle, how many cycles does this loop take to execute?

d. [15] <C.2> Show the timing of this instruction sequence for the 5-stage RISC pipeline with full forwarding and bypassing hardware. Use a pipeline timing chart like that shown in Figure C.5. Assume that the branch is handled by predicting it as taken. If all memory references take 1 cycle, how many cycles does this loop take to execute?

e. [25] <C.2> High-performance processors have very deep pipelines—more than 15 stages. Imagine that you have a 10-stage pipeline in which every stage of the 5-stage pipeline has been split in two. The only catch is that, for data forwarding, data are forwarded from the end of a pair of stages to the beginning of the two stages where they are needed. For example, data are forwarded from the output of the second execute stage to the input of the first execute stage, still causing a 1-cycle delay. Show the timing of this instruction sequence for the 10-stage RISC pipeline with full forwarding and bypassing hardware. Use a pipeline timing chart like that shown in Figure C.5. Assume that the branch is handled by predicting it as taken. If all memory references take 1 cycle, how many cycles does this loop take to execute?

f. [10] <C.2> Assume that in the 5-stage pipeline the longest stage requires 0.8 ns, and the pipeline register delay is 0.1 ns. What is the clock cycle time of the 5-stage pipeline? If the 10-stage pipeline splits all stages in half, what is the cycle time of the 10-stage machine?

g. [15] <C.2> Using your answers from parts (d) and (e), determine the cycles per instruction (CPI) for the loop on a 5-stage pipeline and a 10-stage pipeline. Make sure you count only from when the first instruction reaches the write-back stage to the end. Do not count the start-up of the first instruction. Using the clock cycle time calculated in part (f), calculate the average instruction execute time for each machine.

Problem #4 (10 Points): Page B-60 in H&P5, Problem B.2 a,b

B.2 [15/15] <B.1> For the purpose of this exercise, we assume that we have 512-byte cache with 64-byte blocks. We will also assume that the main memory is 2 KB large. We can regard the memory as an array of 64-byte blocks: M0, M1, ..., M31. Figure B.30 sketches the memory blocks that can reside in different cache blocks if the cache was fully associative.

a. [15] <B.1> Show the contents of the table if cache is organized as a direct-mapped cache.

b. [15] <B.1> Repeat part (a) with the cache organized as a four-way set associative cache.

Cache block	Set	Way	Memory blocks that can reside in cache block
0	0	0	M0, M1, M2, ..., M31
1	0	1	M0, M1, M2, ..., M31
2	0	2	M0, M1, M2, ..., M31
3	0	3	M0, M1, M2, ..., M31
4	0	4	M0, M1, M2, ..., M31
5	0	5	M0, M1, M2, ..., M31
6	0	6	M0, M1, M2, ..., M31
7	0	7	M0, M1, M2, ..., M31

Figure B.30 Memory blocks that can reside in cache block.

Problem #7 (20 Points): Page C-85 in H&P5, Problem C.6 a,b,c,d,e

For problem 7: assume that only sources can use the new addressing mode.

C.6 [12/13/13/15/15] <C.1, C.2, C.3> We will now add support for register-memory ALU operations to the classic five-stage RISC pipeline. To offset this increase in complexity, *all* memory addressing will be restricted to register indirect (i.e., all addresses are simply a value held in a register; no offset or displacement may be added to the register value). For example, the register-memory instruction `ADD R4, R5, (R1)` means add the contents of register R5 to the contents of the memory location with address equal to the value in register R1 and put the sum in register R4. Register-register ALU operations are unchanged. The following items apply to the integer RISC pipeline:

a. [12] <C.1> List a rearranged order of the five traditional stages of the RISC pipeline that will support register-memory operations implemented exclusively by register indirect addressing.

b. [13] <C.2, C.3> Describe what new forwarding paths are needed for the rearranged pipeline by stating the source, destination, and information transferred on each needed new path.

c. [13] <C.2, C.3> For the reordered stages of the RISC pipeline, what new data hazards are created by this addressing mode? Give an instruction sequence illustrating each new hazard.

d. [15] <C.3> List all of the ways that the RISC pipeline with register-memory ALU operations can have a different instruction count for a given program than the original RISC pipeline. Give a pair of specific instruction sequences, one for the original pipeline and one for the rearranged pipeline, to illustrate each way.

e. [15] <C.3> Assume that all instructions take 1 clock cycle per stage. List all of the ways that the register-memory RISC can have a different CPI for a given program as compared to the original RISC pipeline.

Problem #8 (10 Points): Using graph B.9 on page H&P5 B-25 and table B.8 on page B-24. Which has a lower miss rate, a 256KB direct mapped cache or a 64-KB 8-way cache? Which of the three C's drives the previous result?

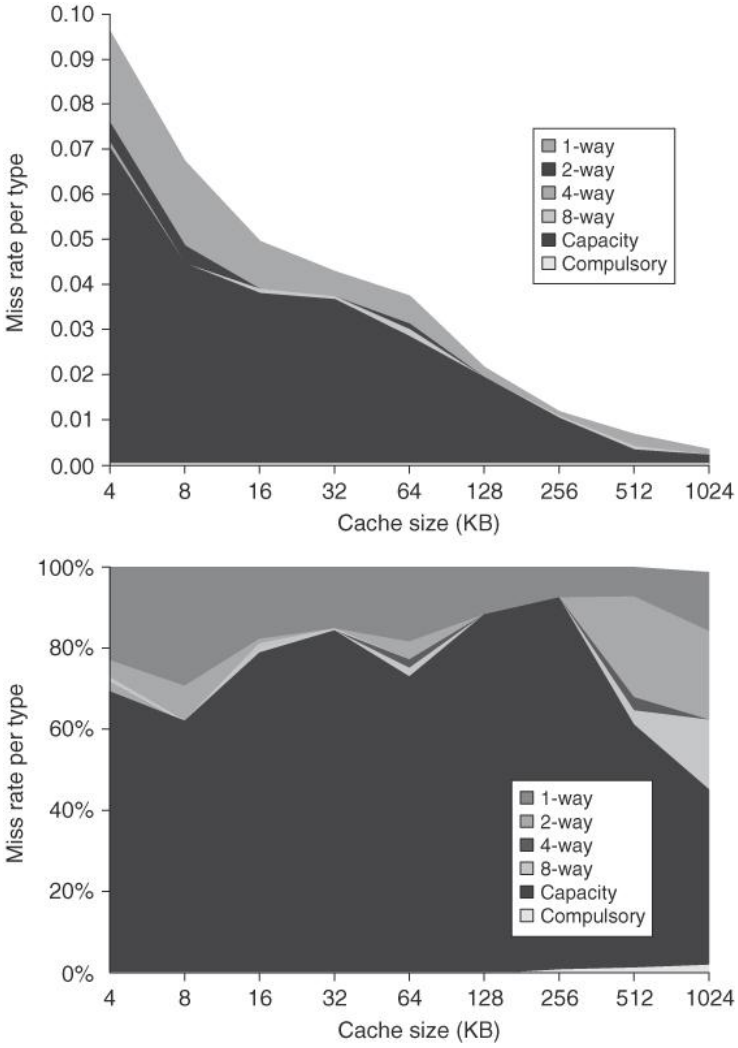

Figure B.9 Total miss rate (top) and distribution of miss rate (bottom) for each size cache according to the three C's for the data in Figure B.8. The top diagram shows the actual data cache miss rates, while the bottom diagram shows the percentage in each category. (Space allows the graphs to show one extra cache size than can fit in Figure B.8.)

Cache size (KB)	Degree associative	Total miss rate	Miss rate components (relative percent) (sum = 100% of total miss rate)					
			Compulsory		Capacity		Conflict	
4	1-way	0.098	0.0001	0.1%	0.070	72%	0.027	28%
4	2-way	0.076	0.0001	0.1%	0.070	93%	0.005	7%
4	4-way	0.071	0.0001	0.1%	0.070	99%	0.001	1%
4	8-way	0.071	0.0001	0.1%	0.070	100%	0.000	0%
8	1-way	0.068	0.0001	0.1%	0.044	65%	0.024	35%
8	2-way	0.049	0.0001	0.1%	0.044	90%	0.005	10%
8	4-way	0.044	0.0001	0.1%	0.044	99%	0.000	1%
8	8-way	0.044	0.0001	0.1%	0.044	100%	0.000	0%
16	1-way	0.049	0.0001	0.1%	0.040	82%	0.009	17%
16	2-way	0.041	0.0001	0.2%	0.040	98%	0.001	2%
16	4-way	0.041	0.0001	0.2%	0.040	99%	0.000	0%
16	8-way	0.041	0.0001	0.2%	0.040	100%	0.000	0%
32	1-way	0.042	0.0001	0.2%	0.037	89%	0.005	11%
32	2-way	0.038	0.0001	0.2%	0.037	99%	0.000	0%
32	4-way	0.037	0.0001	0.2%	0.037	100%	0.000	0%
32	8-way	0.037	0.0001	0.2%	0.037	100%	0.000	0%
64	1-way	0.037	0.0001	0.2%	0.028	77%	0.008	23%
64	2-way	0.031	0.0001	0.2%	0.028	91%	0.003	9%
64	4-way	0.030	0.0001	0.2%	0.028	95%	0.001	4%
64	8-way	0.029	0.0001	0.2%	0.028	97%	0.001	2%
128	1-way	0.021	0.0001	0.3%	0.019	91%	0.002	8%
128	2-way	0.019	0.0001	0.3%	0.019	100%	0.000	0%
128	4-way	0.019	0.0001	0.3%	0.019	100%	0.000	0%
128	8-way	0.019	0.0001	0.3%	0.019	100%	0.000	0%
256	1-way	0.013	0.0001	0.5%	0.012	94%	0.001	6%
256	2-way	0.012	0.0001	0.5%	0.012	99%	0.000	0%
256	4-way	0.012	0.0001	0.5%	0.012	99%	0.000	0%
256	8-way	0.012	0.0001	0.5%	0.012	99%	0.000	0%
512	1-way	0.008	0.0001	0.8%	0.005	66%	0.003	33%
512	2-way	0.007	0.0001	0.9%	0.005	71%	0.002	28%
512	4-way	0.006	0.0001	1.1%	0.005	91%	0.000	8%
512	8-way	0.006	0.0001	1.1%	0.005	95%	0.000	4%

Figure B.8 Total miss rate for each size cache and percentage of each according to the three C's. Compulsory misses are independent of cache size, while capacity misses decrease as capacity increases, and conflict misses decrease as associativity increases. Figure B.9 shows the same information graphically. Note that a direct-mapped cache of size *N* has about the same miss rate as a two-way set-associative cache of size *N*/2 up through 128 K. Caches larger than 128 KB do not prove that rule. Note that the Capacity column is also the fully associative miss rate. Data were collected as in Figure B.4 using LRU replacement.