EE116C/CS151B Homework 8

Problem 1

In this exercise, we will look at the different ways capacity affects overall performance. In general, cache access time is proportional to capacity. Assume that main memory accesses take 70 ns and that memory accesses are 36% of all instructions. The following table shows data for L1 caches attached to each of two processors, P1 and P2.

	L1 size	L1 miss rate	L1 hit time
P1	2KB	8.0%	0.66 ns
P2	4KB	6.0%	0.90 ns

What is the Average Memory Access Time for P1 and P2?

What is the AMAT for P1 with the addition of an L2 cache? Is the AMAT better or worse with the L2 cache?

Problem 2 (a former final problem)

5. ***Why, oh why, must we do TCPI? (40 points)***: We are going to assess branch and cache performance on the pipelined datapath from class – we have full data forwarding. Our peak CPI is 1.0. Assume that 30% of instructions are branches, and that we have a single cycle branch hazard on this processor. Our branch predictor **always** guesses *not taken*. 50% of branches are not taken. Our processor has an instruction cache and data cache – both take a single cycle to access. The instruction cache miss rate is 10% and the data cache miss rate is 30%. The next level of the memory hierarchy is an L2 cache with a miss rate of 20% and an access time of 10 cycles, this is in addition to the L1 cache latency. Main memory has an access time of 80 cycles, this is in addition to the latency of the L1 and L2 caches. 20% of instructions are loads, and stores do not stall the processor on a cache miss. 3/5ths of loads have dependent instructions following them. Our target application executes 1,000,000 instructions. The processor clock runs at 2 GHz.

a. Calculate the average memory access time (AMAT) of the processor:
AMAT: _____ cycles.

b. Calculate TCPI for our target application on our processor.
TCPI: _____ cycles.

c. Suppose $1/6^{th}$ of all branches are procedure calls. Each procedure call (i.e. a jal instruction) in our application also has a return (i.e. a jr instruction). These will all be mispredicted because we always guess not taken. One approach to reducing branch hazards in such a case is to *in-line* the procedure call. The compiler basically takes the instructions in the body of the procedure call and replaces **all** calls to that procedure with these instructions. This means that instead of the code:

```
        add $s0, $s0, $t1
        jal Target
        add $s0, $s0, $t2
        jal Target
        .....
        .....
        .....
        .....
Target: lw $t3, 0 ($s0)
        addi $t3, $t3, 200
        sw $t3, 0 ($s0)
        jr $ra
```

We would have the code:

```
        add $s0, $s0, $t1
        lw $t3, 0 ($s0)
        addi $t3, $t3, 200
        sw $t3, 0 ($s0)
        add $s0, $s0, $t2
        lw $t3, 0 ($s0)
        addi $t3, $t3, 200
        sw $t3, 0 ($s0)
        .....
        .....
        .....
        .....
```

The benefit in this simple example is that we avoid four branches (two jal's and two jr's), but the size of the instruction text segment in memory (i.e. the size of the actual program we are running) has increased. Now, instead of the lw, addi, and sw being in one place in the text segment, they are in two places. This can increase the miss rate of the instruction cache.

Suppose that we try in-lining on our processor. In order for performance to improve, the cost of increasing the instruction cache miss rate must not exceed the benefit of reducing branch hazards. Using TCPI as the CPI in the equation for Execution Time, provide an upper bound on the miss rate of the instruction cache to improve performance when using in-lining. Assume that the L2 cache's miss rate does not change.

c. The instruction cache miss rate must be <= _____