HW #4 Solution (Chapter 5)

5.1.

a) Cache A perform much better than Cache B.

foo:	*beqz*	*r0, bar*	*; branch if r0 = 0*
bar:	*beqz*	*r0, foo*	*; branch if r0 = 0*

Two assumptions: 1. r0 == 0;

2. locations *foo* and *bar* both map onto the same set in both caches.

For cache A, this code only causes two compulsory misses to load the two instructions into the cache. After that point, all accesses hit the cache. For cache B, all the accesses miss the cache. This occurs because a direct-mapped cache can only store one block in each set, yet the program has two active blocks that map to the same set.

b) Cache B perform much better than Cache A.

baz:	*sw*	*0(r1), r0*	*; store r0 to memory*
qux:	*beqz*	*r0, baz*	*; branch if r0 = 0*

Assume: 1. locations *baz* and *qux* and the location pointed to by 0(r1) map to different sets within the caches and are all initially resident;

2. r0 == 0.

Each time *sw* executes on cache A, the data stored is written through to memory. For cache B, *sw* only updates the cache block as the cache is write back; the block is not written to main memory until it is replaced.

c) For the code in a), on cache A, each instruction access hits the cache (we ignore compulsory misses). Thus, the total time required to process cache accesses on cache A is nh_a where n is the number of instructions executed and h_a is the hit time on cache A. On cache B, the code causes a miss on each instruction access. The total time to process all cache accesses is $10h_bn$ where $10h_b$ is the miss time on cache B (misses take 10 times as long as hits on caches). While the miss time represents the total time to process a miss, the miss penalty represents the time beyond the hit time required to process a miss.

If we assume $h_a = h_b$ then cache A runs the code 10 times faster than cache B.

d) Cache B required one read access (which hits the cache) to read a store instruction and one write access(which also hits the cache) to write the data into the cache. This leads to a total time of $2nh_b$. Cache A also requires a read access (which hits the cache) to read a store, but it must write the stored data all the way to main memory because of writing through – which takes five times the hit time. Thus, the time is $n(h_a + 5h_a)$ assuming the write to cache and the write to memory can be overlapped.

If we assume $h_a = h_b$ then cache B runs the code 3 times faster than cache A.

5.5

To evaluate CPU performance, we'd better use the equation of :

$$CPUTime = IC * (CPI_{exe} + \frac{MemroySatllCycle}{Instruction}) * Clk$$

$$CPI_{exe} = f_{load}\, CPI_{load} + f_{store}\, CPI_{store} + f_{other}\, CPI_{other}$$

$$\frac{MemoryStallCycles}{Instruction} = \frac{InstrFetchStallCycles}{Instruction} + \frac{DataStallCycles}{Instruction}$$

$$\frac{StallCycles}{Instruction} = \frac{MemoryAccess}{Instruction} * MissRate * MissPenalty$$

$$\frac{MemoryStallCycles}{Instruction} = \left(100\% R_i P_i\right) + \left[\left(f_{store} + f_{load}\right) R_d P_d\right]$$

where R is miss rate, P is miss penalty, f is frequency.

a) First, $CPI_{exe} = (f_{load} * 1) + (f_{store} * 2) + (f_{other} * 1)$

For memory stalls from instruction fetches, the miss rate $R_i = 0.5\%$, miss penalty $P_i = 50$ cycles. Assume the instruction cache is not written to and therefore does not contain dirty blocks, so there is never write back traffic.

For stalls from data access, the miss rate $R_d = 1\%$, miss penalty P_d depends on cache organization. Write through: $P_d = 50$; write back: $P_d = 50 + (50\% * 50) = 75$.

For a write through cache: $\frac{MemoryStallCycles}{Instruction} = (1\% * 50) + \left[\left(f_{store} + f_{load}\right)(1\% * 50)\right]$

For a write through cache: $\frac{MemoryStallCycles}{Instruction} = (1\% * 50) + \left[\left(f_{store} + f_{load}\right)(1\% * 75)\right]$

Figure 1 summarizes the results for both the write back and write through caches.

Program	Execution CPI	Write Through		Write Back	
		Stalls	Total CPI	Stall	Total CPI
compress	1.05	0.38	1.43	0.44	1.49
eqntott	1.00	0.41	1.41	0.48	1.48
expresso	1.04	0.38	1.42	0.45	1.49
gcc	1.14	0.44	1.58	0.53	1.67
li	1.16	0.49	1.65	0.61	1.77
average	1.08	0.42	1.50	0.50	1.58

Figure 1: Cache performance for seeral SPECint92 programs

b) The write through cache pipelines the writes, so the performance of the write back cache does not change from the results above, for the write through cache, the store term in CPI_{exe} changes.

$CPI_{exe} = (f_{load} * 1) + (f_{store} * 1) + (f_{other} * 1)$

Program	Write Through			Write Back
	Execution CPI	Stalls	Total CPI	Total CPI
compress	1.0	0.38	1.38	1.49
eqntott	1.0	0.41	1.41	1.48
expresso	1.0	0.38	1.38	1.49
gcc	1.0	0.44	1.44	1.67
li	1.0	0.49	1.49	1.77
average	1.0	0.42	1.42	1.58

5.9.

$$\frac{MemoryStallCycles}{Instruction} = \frac{InstrFetchStallCycles}{Instruction} + \frac{DataStallCycles}{Instruction} + \frac{TLBStall}{Instruction}$$

$$\frac{MemoryStallCycles}{Instruction} = \left(R_i P_i\right) + f_{data} R_d P_d + \frac{TLBStall}{Instructoin}$$

a) $R_i = R_d$, $P_i = P_d$, CPI = 1.5 + [RP + (RP * 20%) + 0] = 1.5 + 1.2RP

For a write back cache, the miss penalty depends on the time required to flush and fill a cache block. Block flushes and fills both require 48 clocks to complete (40 for memory latency and 8 to fill the 32-byte block at 4 bytes per clock). While a block fill must always occur on a miss, a block flush only occurs if the block being replaced is dirty, which happens 50% of the time. Thus, the miss penalty is 72 cycles.

b) First we assume the size of the pages mapped by the TLB is a multiple of the block size of the cache, so a cache block can only reside on one page and therefore filling a block from memory can generate at most one TLB access. In addition we can assume that the TLB only slows the processor down when both the TLB and cache miss.

$$\frac{TLBStalls}{Instruction} = \frac{MemoryAccess}{Instruction} * \frac{TLBAccesses}{MemoryAccess} * TLBMissRate * TLBMissPenalty$$

Let f_{data} be the frequency of data access instructions, R_t and P_t be the miss rate and penalty of the TLB, and R_c and R_w be the miss rate and frequency of write backs of the cache, then

$$\frac{TLBStalls}{Instruction} = (1 + f_{data}) * [(R_c(1 + R_w)] * R_t * P_t$$

$R_c(1+R_w)$ corresponds to the number of TLB accesses per memory access. Because the TLB can only potentially slow down the system on a cache moss, we multiply the second term by the miss rate of the cache R_c. On a cache miss, the TLB must translate the address of the block to be read from memory and also may have to translate the address of the block to be flushed to memory if a write back is necessary.

$$\frac{TLBStalls}{Instruction} = (1 + 20\%) * [(R_c(1 + 50\%)] * 0.2\% * 20$$

Cache	Miss Rate	Ideal TLB	Real TLB	
		CPI	Stalls	CPI
16KB Direct	0.029	4.0	0.17	4.2
16KB Two-way	0.022	3.4	0.13	3.5
32KB Direct	0.020	3.2	0.12	3.3

From the results we can see that the TLB increases the CPI by around 5% in each case. Although the penalty for missing the TLB is severe(20 cycles), it does not happen very frequently.

c) The addressing of the caches impacts TLB performance by changing the performance demands required to keep hit time small. In large **physically addressed caches**, some address translation needs to occur **before** the cache access can be completed. As a result, the TLB must be able to provide the translation quickly if hit time is to be kept low (essentially, in this case the TLB and cache are **"in series"**).

For **virtually addressed caches**, the TLB can be accessed **in parallel** with the cache. This reduces the pressure on TLB performance as the cache does not need information generated by the TLB to operate. Because the TLB must be accessed **prior to** any cache access in a physically addressed cache, there is also the danger of additional wasted cycles if the address translation is not in the TLB but the desired data is in the cache. The extra time spent servicing the TLB miss would not be seen in a virtually addressed cache if the desired information is resident in the cache.