Hunter Morera
U79718512
10/17/18
HW2

B1

a. To calculate the average access time for a program that has a miss rate of 3% we use the following equation:

$$AMAT = Hit\ Time + (Miss\ Rate * Miss\ Penalty)$$

We can then substitute in our variables and get the following:

$$AMAT = 1 + (0.03 * 110) = 1 + 3.3 = 4.3\ cycles$$

Therefore the average memory access time for a program that has a miss rate of 3% is 4.3 cycles.

b. To calculate average access time for this program with 1GB array and 64KB cache we use the following equation:

$$AMAT = Hit\ Time + (Miss\ Rate * Miss\ Penalty)$$

We know the hit time, and miss penalty but we need to calculate our miss rate. Since indices are accessed at random and we have at most 64KB of the array in cache at a time, and since 1GB = 1,000,000KB, we can calculate miss rate as follows:

$$Miss\ Rate = 100 - (\frac{64}{1000000} * 100) \approx 99.986\% = 0.99986$$

Now that we calculated our miss rate we can substitute in all of our variables to get the following:

$$AMAT = 1 + (0.99986 * 110) \approx 110\ cycles$$

Therefore the average memory access time for this program with an array of 1GB size and a cache of 64KB is 110 cycles.

c. If we repeat the previous problem with the cache disabled then all of our memory accesses would have to go straight to main memory with a penalty of 105 cycles. Since 100% of memory accesses would do that with the cache disabled our AMAT = 105 cycles without the cache. Compare that to the previous calculation with the 64KB cache, we can see that because such a small portion of the array is held in cache, and being that no locality is being used in accessing elements of the array, that it is actually more expensive to use the cache in this instance. If we were accessing the array with spatial locality in mind, say inside a sequential for loop, then we would reduce our misses significantly because we would only need to access Main Memory a fraction of the time of the previous example. This problem shows how the concept of locality can greatly affect the benefits of having cache memory.

d. In order to calculate the percent of misses at which a cache would be disadvantageous to have a cache we would use the following equation.

$$No\ Cache\ Access\ Time = (Hit\ Time + (Miss\ Rate * Miss\ Penalty)$$

We can then substitute in our variables to get:

$$105 = (1 + (MR * 110)$$

We can then solve for MR and get:

$$104 = MR * 110$$

Hunter Morera
U79718512
10/17/18
HW2

$\frac{104}{110} = MR$

MR = 94.5%

Therefore if the miss rate is larger than 94.5%, it is no longer worth having a cache, the system would run faster if every access was from main memory.

B2

a. The contents of the table if we used a fully associative cache would be as follows:

Cache Block	Set	Way	Possible Mem Blocks
0	0	0	M1,M2….M31
1	0	1	M1,M2….M31
2	0	2	M1,M2….M31
3	0	3	M1,M2….M31
4	0	4	M1,M2….M31
5	0	5	AM1,M2….M31
6	0	6	AM1,M2….M31
7	0	7	M1,M2….M31

b. The contents of the table if we used 4-way associative cache would be as follows:

Cache Block	Set	Way	Possible Mem Blocks
0-3	0	0	M0,M2,....M30
4-7	0	1	M1,M3,....M31

B5

a. To Calculate the AMAT for instruction access we use the following equation:

$$AMAT = HIT\ Time(L1) + Miss\ Rate(L1) * Miss\ Penalty(L1)$$

And the miss penalty of L1 is defined as follows:

$$Miss\ Penalty(L1) = Hit\ Time(L2) + Miss\ Rate(L2) * Miss\ Penalty(L2)$$

We can then begin to substitute in our numbers to calculate our Miss Penalty of L1:

$$Hit\ Time(L2) = 2 * bus\ speed + access\ time = 2 * 3.75ns + 15ns = 22.5ns$$

Hunter Morera
U79718512
10/17/18
HW2

$Miss\ Rate(L2) = 20\%$

$Miss\ Penalty(L2) = 4 * bus\ speed + access\ time = 4 * 7.52 + 60ns = 90.08ns + dirty\ bit\ replacement$

$Miss\ Penalty(L2) = 90ns + 0.5 * 90ns = 135ns$

$Miss\ Penalty(L1) = 22.5ns + (.2 * 135ns) = 49.5ns$

$AMAT = 0 + (0.02 * 49.5) = 0.99ns$

Therefore the AMAT of instruction access is 0.99ns

b. **a**. To calculate the AMAT for data reads access we use the following equation:

$AMAT = HIT\ Time(L1) + Miss\ Rate(L1) * Miss\ Penalty(L1)$

And the miss penalty of L1 is defined as follows:

$Miss\ Penalty(L1) = Hit\ Time(L2) + Miss\ Rate(L2) * Miss\ Penalty(L2)$

We can then begin to substitute in our numbers to calculate our Miss Penalty of L1:

$Hit\ Time(L2) = bus\ speed + access\ time = 3.75ns + 15ns = 18.75ns$

$Miss\ Rate(L2) = 20\%$

$Miss\ Penalty(L2) = 4 * bus\ speed + access\ time = 4 * 7.52ns + 60ns = 90.08ns + dirty\ bit\ replacement$

$Miss\ Penalty(L2) = 90ns + 0.5 * 90ns = 135ns$

$Miss\ Penalty(L1) = 18.75ns + (.2 * 135ns) = 45.75ns$

$AMAT = 0 + (0.05 * 45.75) = 2.28ns$

Therefore the AMAT of data read access is 2.28ns

c. To calculate the AMAT for data writes we use the following equation:

$AMAT = Data\ cache\ stall\ percent * Miss\ penalty\ of\ L1$

$Miss\ Penalty(L1) = Hit\ Time(L2) + Miss\ Rate(L2) * Miss\ Penalty(L2)$

We can then calculate the Miss penalty of L2 as follows:

$Hit\ Time(L2) = 3.75ns + 15ns = 18.75ns$

$Miss\ Penalty(L2) = 4 * bus\ speed + access\ time = 4 * 7.52ns + 60ns = 90.08ns$

$Miss\ Penalty(L2) = 90ns + 0.5 * 90ns = 135ns$

$Miss\ Penalty(L1) = 18.75ns + 135ns = 153.75ns$

$AMAT = 0.05(153.75) = 7.69ns$

Therefore the AMAT of data writes is 7.69ns

d. To calculate the overall CPI we must first convert our previous answers to cycles:

$Instruction\ Access = 0.99ns * 1.1 = 1.089\ Clock\ cycles$

$Data\ Read\ Access = 2.28ns * 1.1 = 2.508\ Clock\ cycles$

$Data\ Write = 7.69ns * 1.1 = 8.459\ Clock\ cycles$

We can now calculate the overall CPI as follows:

$Overall\ CPI = Base\ CPI + (0.2 * Read\ cycles) + (0.1 * Write\ cycles) + instruction\ cycles$

We can then substitute in our variables:

$Overall\ CPI = 1.35 + (0.2 * 2.508) + (0.1 * 8.459) + 1.089 = 3.7865 \approx 3.79$

Hunter Morera
U79718512
10/17/18
HW2

Therefore the overall CPI is 3.79

B9

Let's assume the sequence of memory accesses were: 0,3,4,0,3.
We will start out by showing how these accesses are handled in a direct mapped cache with two blocks of memory, each holds one word.

Memory Accesses	Hit or Miss	0	1
0	Miss	Mem(0)	
3	Miss	Mem(0)	Mem(3)
4	Miss	Mem(4)	Mem(3)
0	Miss	Mem(0)	Mem(3)
3	Hit	Mem(0)	Mem(3)

Now we will show how these accesses are handled in a two-way set associative cache with two blocks of memory where each holds one word and implements a LRU replacement policy.

Memory Access	Hit or Miss	Set 0	
0	Miss	Mem(0)	
3	Miss	Mem(0)	Mem(3)
4	Miss	Mem(4)	Mem(3)
0	Miss	Mem(4)	Mem(0)
3	Miss	Mem(3)	Mem(0)

As we can see in our direct mapped cache we had 4 misses and in our 2-way set associative cache we had 5 misses. Therefore we can see there are some memory access sequences which cause a 2-way set associative cache to have more misses than a direct mapped cache.

2.18

a. We can begin by calculating the AMAT of the current cache setup:

$$AMAT_{current\ cache} = Hit\ Time + (Miss\ Rate * Miss\ Penalty)$$

We can then substitute in our variables and get:

Hunter Morera
U79718512
10/17/18
HW2

$AMAT_{current\ cache} = 3 + (0.33 * 20) = 3 + (0.066) = 3.066\ cycles$

Now we can calculate the AMAT of the new proposed design:

$AMAT_{way-pred} = (pred\ accuracy * Hit\ Time) + (1 - pred\ accuracy * 1 + Hit\ Time)$

We can then substitute in our variables and get:

$AMAT_{way-pred} = (80\% * 2\ cycles) + (1 - 80\% * 1 + 2\ cycles)$

$AMAT_{way-pred} = (0.8 * 2\ cycles) + (0.2 * 3\ cycles)$

$AMAT_{way-pred} = (1.6) + (0.6) = 2.2\ cycles$

Therefore the AMAT of the current cache is appx 3.066 cycles, and the AMAT of the proposed way predicting cache is 2.2 cycles.

c. To solve this problem we need to start with our original cache AMAT from part a.

$AMAT_{current\ cache} = 3 + (0.33\% * 20) = 3 + (0.066) = 3.066\ cycles$

Then we need to determine how many cycles this new cache setup will take:

$AMAT_{way-pred} = Hit\ Time + (Miss\ Rate * Miss\ Penalty)$

We can substitute in our variables and get:

$AMAT_{way-pred} = (0.8 * 2) + (0.2 * 15) + (0.33 * 20) = 4.66\ cycles$

We calculate the difference as follows:

$AMAT_{current\ cache} / AMAT_{way-pred} = 3.066 / 5 = 0.61\ cycles$

Therefore the change in access time between the original cache and way predicting data cache is 0.61 cycles.

2.20

a. Lets first calculate the cycles needed to service L2 cache with critical word first and early restart, it is simply the cycles needed to receive the first block from the memory controller:

$L2\ Miss\ Service_{w/\ critical\ word\ first} = 120\ cycles$

Now lets calculate the time it would take without word first and early start:

$L2\ Miss\ Service_{w/o\ critical\ word\ first} = first\ block\ cycles + (3 * 16\ cycles)$

$L2\ Miss\ Service_{w/o\ critical\ word\ first} = 120 + (3 * 16\ cycles) = 168\ cycles$

Therefore with the critical word first and early restart it would take 120 cycles to service a miss, and without it would take 168 cycles to service a miss.

b. To say which cache the critical word first and early start would benefit more we need to know which cache contributes the most misses and what amount of reduction in servicing those misses does the critical word first and early start provide. Assuming that it provides a significant reduction cycles then we would want to apply it to the cache which is responsible for the rise in AMAT. If that is the L1 then we should apply it to the L1, if its the L2 then it should be applied to the L2.

Hunter Morera
U79718512
10/17/18
HW2

2.21

a. Each write buffer should be 16 bytes wide so that it is the same as the L2 write data bus.

b. To calculate the difference in speed between the a merging write buffer and non merging write buffer we must calculate the speed of each.

$$Speed_{non-merging} = \frac{16\ bytes}{4\ cycles} = 4\ bytes/cycle$$

Now we calculate the speed of a merging write buffer

$$Speed_{merging} = \frac{64\ bits}{1\ cycle} = \frac{8\ bytes}{1\ cycle} = 8\ bytes/cycle$$

We can calculate the speed up as follows:

$$Speed - Up = \frac{Speed\ of\ merging}{Speed\ of\ non-merging} = \frac{8}{4} = 2\ times\ speed\ up$$

Therefore we can see that a speedup of 2 can be expected if we use a merging write buffer instead of a non-merging buffer.

c. In a non-blocking cache, the cache will continue to service requests even if there is a cache miss, this means that writes in the write buffer can go ahead and be processed even if there is a miss, this results in fewer entries. If you have a blocking cache, this means that if you have a miss the cache will not allow any more progress until that miss is handled and the data request is satisfied, however this has no impact on the number of required write buffer entries.