EE 180 Homework 3

Sean William Konz, Schuyler Anne Tilney-Volk

TOTAL POINTS

96 / 100

QUESTION 1

1 10 pts

1.1 a) **3 / 3**

✓ **- 0 pts Correct**

- 3 pts wrong answer

1.2 b) **3 / 3**

✓ **- 0 pts Correct**

- 2 pts wrong answer

1.3 c) **4 / 4**

✓ **- 0 pts Correct**

- 0.5 pts final answer wrong

- 0.5 pts minor error

QUESTION 2

2 2 **10 / 12**

- 0 pts Correct

- 7 pts 7 cells are wrong

- 3 pts 3 cells are missing

- 1 pts 1 cell is wrong

✓ **- 2 pts 2 cells are wrong**

- 4 pts 4 cells are wrong

QUESTION 3

3 20 pts

3.1 a) **4 / 4**

✓ **- 0 pts Correct**

- 3 pts wrong answers

- 0 pts missing_L2_dataread

- 1 pts minor error not included dirtyrate_L2

- 2 pts calculation or value error

- 1 pts minor calculation error

3.2 b) **4 / 4**

✓ **- 0 pts Correct**

- 2 pts wrong calculation

- 1 pts error with values

- 0.5 pts error considering the writehittime

3.3 c) **4 / 4**

✓ **- 0 pts Correct**

- 1 pts error with numbers or wrong calculation

- 2 pts wrong calculation

3.4 d) **4 / 4**

✓ **- 0 pts Correct**

- 1 pts correct approach wrong answer based on earlier parts

- 1 pts wrong numbers

- 2 pts wrong calculation

3.5 e) **4 / 4**

✓ **- 0 pts Correct**

- 0.5 pts wrong calculation based on calculated part d

QUESTION 4

4 28 pts

4.1 a) **5 / 5**

✓ **- 0 pts Correct**

4.2 b) **5 / 5**

✓ **- 0 pts Correct**

- 1 pts Provide more explanation

- 2 pts Click here to replace this description.

4.3 c) **5 / 5**

✓ **- 0 pts Correct**

- **2 pts** provide detailed explanation
- **0 pts** Click here to replace this description.

4.4 d) **4 / 4**

✓ **- 0 pts Correct**

- **1 pts** explain in detail

4.5 e) **4 / 4**

✓ **- 0 pts Correct**

- **1 pts** give more explanation

4.6 f) **5 / 5**

✓ **- 0 pts Correct**

QUESTION 5

5 18 pts

5.1 a) **6 / 6**

✓ **- 0 pts Correct**

- **1 pts** some values are wrong
- **2 pts** more than 5 values are wrong

5.2 b) **4 / 4**

✓ **- 0 pts Correct**

- **0 pts** see solution

5.3 c) **6 / 6**

✓ **- 0 pts Correct**

5.4 d) **2 / 2**

✓ **- 0 pts Correct**

- **1 pts** provide proper reason

QUESTION 6

6 6 **10 / 12**

- **0 pts** Correct
- **3 pts** 3 Cells are wrong
- **4 pts** 4 Cell are wrong
- **1 pts** 1 cell is wrong

✓ **- 2 pts 2 cells are wrong**

- **10 pts** 10 cells are wrong

EE180 - HW3

Sean Konz

(1) a) Offset determines number of words in a block;

$2^{5-2} = 8$ words

b) Index determines number of entries (# of locations to be looked up)

$2^8 = 256$ entries

c) Required number of bits in the cache is determined by the write back policy since different policies require different number of bits, to determine entry validity and write status;

Always have a 19 bit tag, and a valid bit

i) Assuming write through, no additional information is needed;

$$\frac{2^9 + 20}{2^9} = 1 + \frac{20}{2^8} = 1.078125$$

Assuming write back, we need an additional "dirty" bit!

$$\frac{2^9 + 21}{2^9} = 1 + \frac{21}{2^8} = 1.08203$$

1.1 a) **3 / 3**

✓ **- 0 pts** **Correct**

- 3 pts wrong answer

EE180 - HW3

Sean Konz

(1) a) Offset determines number of words in a block;

$2^{5-2} = 8$ words

b) Index determines number of entries (# of locations to be looked up)

$2^8 = 256$ entries

c) Required number of bits in the cache is determined by the write back policy since different policies require different number of bits, to determine entry validity and write status;

Always have a 19 bit tag, and a valid bit

i) Assuming write through, no additional information is needed;

$$\frac{2^9 + 20}{2^9} = 1 + \frac{20}{2^8} = 1.078125$$

Assuming write back, we need an additional "dirty" bit!

$$\frac{2^9 + 21}{2^9} = 1 + \frac{21}{2^8} = 1.08203$$

1.2 b) **3 / 3**

✓ **- 0 pts** **Correct**

- 2 pts wrong answer

EE180 - HW3

Sean Konz

(1) a) Offset determines number of words in a block;

$2^{5-2} = 8$ words

b) Index determines number of entries (# of locations to be looked up)

$2^8 = 256$ entries

c) Required number of bits in the cache is determined by the write back policy since different policies require different number of bits, to determine entry validity and write status;

Always have a 19 bit tag, and a valid bit

i) Assuming write through, no additional information is needed;

$$\frac{2^9 + 20}{2^9} = 1 + \frac{20}{2^8} = 1.078125$$

Assuming write back, we need an additional "dirty" bit!

$$\frac{2^9 + 21}{2^9} = 1 + \frac{21}{2^8} = 1.08203$$

1.3 c) **4 / 4**

✓ **- 0 pts** **Correct**

- 0.5 pts final answer wrong

- 0.5 pts minor error

(2)

	Compulsory Misses	Conflict Misses	Capacity Miss
Double Associativity	**No impact** Compulsory misses are not impacted by cache design.	**Decrease** more blocks at each set, less need to evict blocks that map to the same index	**no impact** Doesn't change actual # of blocks available
Halving the line size	**increase** Halving line size also halves the amount of data fetched on each miss. Decreasing line size decreases locality impact.	**Increase** Fewer offset bits, so TAG is larger, so more possibilities mapped to each block	**increase** Halving line size and keeping everything else constant will decrease the capacity of the cache
Double number of sets	**No impact** line size + capacity are constant	**Increase** Doubling # of sets and keeping capacity constant will decrease associativity	**No impact** Capacity is constant
Adding Prefetching	**Decrease** The data required will be placed in the cache before it's needed, avoiding a miss	**Increase** More data being fetched increases the likelyhood of a conflict. If the prefetch guess is not perfect, data will be loaded or overwritten without any utilization	**Decrease** Prefetching makes the apparent size of the cache larger since data is available when requested since prefetch placed it in the cache ahead of time

2 2 **10 / 12**

- **0 pts** Correct
- **7 pts** 7 cells are wrong
- **3 pts** 3 cells are missing
- **1 pts** 1 cell is wrong

✓ **- 2 pts** **2 cells are wrong**

- **4 pts** 4 cells are wrong

(3) Clock = 5 GHz Clock Time = CT = .2ns Hit Time = HT Miss Rate = MR

$HT_{L1} = .2ns$ $MR_{L1D} = .11$

$HT_{L2} = 25ns$ $MR_{L2} = .15$ .165% of 15% go to main memory

$MR_{L1I} = .06$ $P_{Load} = .2$ $P_{Store} = .12$

a) $AMAT_{DataRead} = HT_{L1} + MR_{L1D}(HT_{L2} + .15(.65(80ns) + 80ns))$

$= .2ns + .11(25ns + .15(.65(80ns) + 80ns))$

$= 5.128ns$

b) If Write Hit → check if hit (1 cycle) + perform write (1 cycle)

if Write miss → same as read miss

$AMAT_{DataWrite} = 2\cdot(.2ns) + MR_{L1D}(HT_{L2} + .15(.65(80ns) + 80ns))$

$= 2\cdot(2ns) + .11(25ns + .15(.65(80ns) + 80ns))$

$= 5.328ns$

c) $AMAT_{InstRead} = HT_{L1} + MR_{L1I}(HT_{L2} + .15(.65(80ns) + 80ns))$

$= .2ns + .06(25ns + .15(.65(80ns) + 80ns))$

$= 2.88ns$

d) $CPI_{overall} = CPI_{Init} + Clock(AMAT_{InstRead} + P_{Load}AMAT_{DR} + P_{Store}AMAT_{DW})$

$= 1.3 + 5\times10^9(2.88ns + (.2)(5.128ns) + (.12)(5.328ns))$

$= 24.0248$

e) $Time = \frac{Inst \cdot CPI}{Clock} = \frac{20\times10^6 \cdot 24.0248}{5\times10^9} = .0960992s$

3.1 a) **4 / 4**

✓ **- 0 pts** **Correct**

- 3 pts wrong answers

- 0 pts missing_L2_dataread

- 1 pts minor error not included dirtyrate_L2

- 2 pts calculation or value error

- 1 pts minor calculation error

(3) Clock = 5 GHz Clock Time = CT = .2ns Hit Time = HT Miss Rate = MR

$HT_{L1} = .2ns$ $MR_{L1D} = .11$

$HT_{L2} = 25ns$ $MR_{L2} = .15$ .165% of 15% go to main memory

$MR_{L1I} = .06$ $P_{Load} = .2$ $P_{Store} = .12$

a) $AMAT_{DataRead} = HT_{L1} + MR_{L1D}(HT_{L2} + .15(.65(80ns) + 80ns))$

$= .2ns + .11(25ns + .15(.65(80ns) + 80ns))$

$= 5.128ns$

b) If Write Hit → check if hit (1 cycle) + perform write (1 cycle)

if Write miss → same as read miss

$AMAT_{DataWrite} = 2\cdot(.2ns) + MR_{L1D}(HT_{L2} + .15(.65(80ns) + 80ns))$

$= 2\cdot(2ns) + .11(25ns + .15(.65(80ns) + 80ns))$

$= 5.328ns$

c) $AMAT_{InstRead} = HT_{L1} + MR_{L1I}(HT_{L2} + .15(.65(80ns) + 80ns))$

$= .2ns + .06(25ns + .15(.65(80ns) + 80ns))$

$= 2.88ns$

d) $CPI_{overall} = CPI_{Init} + Clock(AMAT_{InstRead} + P_{Load}AMAT_{DR} + P_{Store}AMAT_{DW})$

$= 1.3 + 5\times10^9(2.88ns + (.2)(5.128ns) + (.12)(5.328ns))$

$= 24.0248$

e) $Time = \frac{Inst \cdot CPI}{Clock} = \frac{20\times10^6 \cdot 24.0248}{5\times10^9} = .0960992s$

3.2 b) 4 / 4

✓ **- 0 pts** **Correct**

- 2 pts wrong calculation

- 1 pts error with values

- 0.5 pts error considering the writehittime

(3) Clock = 5 GHz Clock Time = CT = .2ns Hit Time = HT Miss Rate = MR

$HT_{L1} = .2ns$ $MR_{L1D} = .11$

$HT_{L2} = 25ns$ $MR_{L2} = .15$ .165% of 15% go to main memory

$MR_{L1I} = .06$ $P_{Load} = .2$ $P_{Store} = .12$

a) $AMAT_{DataRead} = HT_{L1} + MR_{L1D}(HT_{L2} + .15(.65(80ns) + 80ns))$

$= .2ns + .11(25ns + .15(.65(80ns) + 80ns))$

$= 5.128ns$

b) If Write Hit → check if hit (1 cycle) + perform write (1 cycle)

if Write miss → same as read miss

$AMAT_{DataWrite} = 2\cdot(.2ns) + MR_{L1D}(HT_{L2} + .15(.65(80ns) + 80ns))$

$= 2\cdot(2ns) + .11(25ns + .15(.65(80ns) + 80ns))$

$= 5.328ns$

c) $AMAT_{InstRead} = HT_{L1} + MR_{L1I}(HT_{L2} + .15(.65(80ns) + 80ns))$

$= .2ns + .06(25ns + .15(.65(80ns) + 80ns))$

$= 2.88ns$

d) $CPI_{overall} = CPI_{Init} + Clock(AMAT_{InstRead} + P_{Load}AMAT_{DR} + P_{Store}AMAT_{DW})$

$= 1.3 + 5\times10^9(2.88ns + (.2)(5.128ns) + (.12)(5.328ns))$

$= 24.0248$

e) $Time = \frac{Inst \cdot CPI}{Clock} = \frac{20\times10^6 \cdot 24.0248}{5\times10^9} = .0960992s$

3.3 c) **4 / 4**

✓ **- 0 pts** **Correct**

- 1 pts error with numbers or wrong calculation

- 2 pts wrong calculation

(3) Clock = 5 GHz Clock Time = CT = .2ns Hit Time = HT Miss Rate = MR

$HT_{L1} = .2ns$ $MR_{L1D} = .11$

$HT_{L2} = 25ns$ $MR_{L2} = .15$ .165% of 15% go to main memory

$MR_{L1I} = .06$ $P_{Load} = .2$ $P_{Store} = .12$

a) $AMAT_{DataRead} = HT_{L1} + MR_{L1D}(HT_{L2} + .15(.65(80ns) + 80ns))$

$= .2ns + .11(25ns + .15(.65(80ns) + 80ns))$

$= 5.128ns$

b) If Write Hit → check if hit (1 cycle) + perform write (1 cycle)

if Write miss → same as read miss

$AMAT_{DataWrite} = 2\cdot(.2ns) + MR_{L1D}(HT_{L2} + .15(.65(80ns) + 80ns))$

$= 2\cdot(2ns) + .11(25ns + .15(.65(80ns) + 80ns))$

$= 5.328ns$

c) $AMAT_{InstRead} = HT_{L1} + MR_{L1I}(HT_{L2} + .15(.65(80ns) + 80ns))$

$= .2ns + .06(25ns + .15(.65(80ns) + 80ns))$

$= 2.88ns$

d) $CPI_{overall} = CPI_{Init} + Clock(AMAT_{InstRead} + P_{Load}AMAT_{DR} + P_{Store}AMAT_{DW})$

$= 1.3 + 5\times10^9(2.88ns + (.2)(5.128ns) + (.12)(5.328ns))$

$= 24.0248$

e) $Time = \frac{Inst \cdot CPI}{Clock} = \frac{20\times10^6 \cdot 24.0248}{5\times10^9} = .0960992s$

3.4 d) **4 / 4**

✓ **- 0 pts** **Correct**

- 1 pts correct approach wrong answer based on earlier parts

- 1 pts wrong numbers

- 2 pts wrong calculation

(3) Clock = 5 GHz Clock Time = CT = .2ns Hit Time = HT Miss Rate = MR

$HT_{L1} = .2ns$ $MR_{L1D} = .11$

$HT_{L2} = 25ns$ $MR_{L2} = .15$ .165% of 15% go to main memory

$MR_{L1I} = .06$ $P_{Load} = .2$ $P_{Store} = .12$

a) $AMAT_{DataRead} = HT_{L1} + MR_{L1D}(HT_{L2} + .15(.65(80ns) + 80ns))$

$= .2ns + .11(25ns + .15(.65(80ns) + 80ns))$

$= 5.128ns$

b) If Write Hit → check if hit (1 cycle) + perform write (1 cycle)

if Write miss → same as read miss

$AMAT_{DataWrite} = 2\cdot(.2ns) + MR_{L1D}(HT_{L2} + .15(.65(80ns) + 80ns))$

$= 2\cdot(2ns) + .11(25ns + .15(.65(80ns) + 80ns))$

$= 5.328ns$

c) $AMAT_{InstRead} = HT_{L1} + MR_{L1I}(HT_{L2} + .15(.65(80ns) + 80ns))$

$= .2ns + .06(25ns + .15(.65(80ns) + 80ns))$

$= 2.88ns$

d) $CPI_{overall} = CPI_{Init} + Clock(AMAT_{InstRead} + P_{Load}AMAT_{DR} + P_{Store}AMAT_{DW})$

$= 1.3 + 5\times10^9(2.88ns + (.2)(5.128ns) + (.12)(5.328ns))$

$= 24.0248$

e) $Time = \frac{Inst \cdot CPI}{Clock} = \frac{20\times10^6 \cdot 24.0248}{5\times10^9} = .0960992s$

3.5 e) **4 / 4**

✓ **- 0 pts** **Correct**

- 0.5 pts wrong calculation based on calculated part d

④ a) Since he runs the inner loop 1000 times, and compulsory misses only occur on the first iteration, they can essentially be ignored.

b) The array currently completely fits into the cache, but at some point, as Ben continues doubling the size, the array will fit exactly into the cache since both the array size and the cache size are powers of two. This array size will still yield an acceptable AMAT since there are no main memory accesses. If Ben continues to increase the size of the array, the array size will then be double the size of the cache. This means that for every inner loop iteration, only half the array will fit in the cache at any one time, and the half in the cache will be the opposite half of the array than is being opperated on. at any given time. Therefore, we access main memory on each execution of the inner loop, so the AMAT time will jump to Main memory access time (1000ns). So when Ben sees the AMAT jump to 2000ns, he knows that the current array size is twice the size of the cache size.

c) If the block size is 4, Ben should expect AMAT to be 2. Main Memory (2.100ns). If the block size is greater than 4, then the AMAT will fall somewhere between L1 Hit time and Main Memory access time. When we read a new block, we'll have to access main memory, but within a block, we only access the L1. If the block size is S, and $S \geq 4$ bytes & S is a multiple of 2, then reading a new block from memory causes us to read S/4 elements of the array, so the next S/4 - 1 elements accessed after a block fetch will have L1 hit time for accessing

4.1 a) **5 / 5**

✓ **- 0 pts** Correct

④ a) Since he runs the inner loop 1000 times, and compulsory misses only occur on the first iteration, they can essentially be ignored.

b) The array currently completely fits into the cache, but at some point, as Ben continues doubling the size, the array will fit exactly into the cache since both the array size and the cache size are powers of two. This array size will still yield an acceptable AMAT since there are no main memory accesses. If Ben continues to increase the size of the array, the array size will then be double the size of the cache. This means that for every inner loop iteration, only half the array will fit in the cache at any one time, and the half in the cache will be the opposite half of the array than is being opperated on. at any given time. Therefore, we access main memory on each execution of the inner loop, so the AMAT time will jump to Main memory access time (1000ns). So when Ben sees the AMAT jump to 2000ns, he knows that the current array size is twice the size of the cache size.

c) If the block size is 4, Ben should expect AMAT to be 2. Main Memory (2.100ns). If the block size is greater than 4, then the AMAT will fall somewhere between L1 Hit time and Main Memory access time. When we read a new block, we'll have to access main memory, but within a block, we only access the L1. If the block size is S, and $S \geq 4$ bytes & S is a multiple of 2, then reading a new block from memory causes us to read S/4 elements of the array, so the next S/4 - 1 elements accessed after a block fetch will have L1 hit time for accessing

4.2 b) **5 / 5**

✓ **- 0 pts** **Correct**

- 1 pts Provide more explanation

- 2 pts Click here to replace this description.

④ a) Since he runs the inner loop 1000 times, and compulsory misses only occur on the first iteration, they can essentially be ignored.

b) The array currently completely fits into the cache, but at some point, as Ben continues doubling the size, the array will fit exactly into the cache since both the array size and the cache size are powers of two. This array size will still yield an acceptable AMAT since there are no main memory accesses. If Ben continues to increase the size of the array, the array size will then be double the size of the cache. This means that for every inner loop iteration, only half the array will fit in the cache at any one time, and the half in the cache will be the opposite half of the array than is being opperated on. at any given time. Therefore, we access main memory on each execution of the inner loop, so the AMAT time will jump to Main memory access time (1000ns). So when Ben sees the AMAT jump to 2000ns, he knows that the current array size is twice the size of the cache size.

c) If the block size is 4, Ben should expect AMAT to be 2. Main Memory (2.100ns). If the block size is greater than 4, then the AMAT will fall somewhere between L1 Hit time and Main Memory access time. When we read a new block, we'll have to access main memory, but within a block, we only access the L1. If the block size is S, and $S \geq 4$ bytes & S is a multiple of 2, then reading a new block from memory causes us to read S/4 elements of the array, so the next S/4 - 1 elements accessed after a block fetch will have L1 hit time for accessing

4.3 c) **5 / 5**

✓ **- 0 pts** **Correct**

- 2 pts provide detailed explanation

- 0 pts Click here to replace this description.

d) If the cache is DM, the AMAT should be equal to the main memory access time since, if the cache is DM, the 0th and $(16 \cdot 1024)^{th}$ element would map to the same block, leading to an eviction on each increment of the inner loop

e) If the cache is two-way set associative, the AMAT should be approximately the L1 access time since when both elements are mapped to the same set, there are 2 ways to store either element, so on each execution of the inner loop, the values don't need to be fetched from memory again since they aren't evicting one another. On each execution of the inner loop.

f) We can use a similar approach to the one utilized in previous stages to accomplish this. With an ARRAY_SIZE = 64·1024 (4x cache size) and STEP_SIZE = 16·1024, we'll be accessing 4 elements of the array, all of which would be mapped to the same set in the cache. In a 4-way associative cache, all 4 elements would be able to fit into the set, so the AMAT should be about L1 access time. If the cache is a 2-way associative cache, only 2 elements would be able to be stored at once, which means that on every iteration of the inner loop we will have to access main memory, so AMAT will be Main memory access time.

4.4 d) **4 / 4**

✓ **- 0 pts** **Correct**

- 1 pts explain in detail

d) If the cache is DM, the AMAT should be equal to the main memory access time since, if the cache is DM, the 0th and $(16 \cdot 1024)^{th}$ element would map to the same block, leading to an eviction on each increment of the inner loop

e) If the cache is two-way set associative, the AMAT should be approximately the L1 access time since when both elements are mapped to the same set, there are 2 ways to store either element, so on each execution of the inner loop, the values don't need to be fetched from memory again since they aren't evicting one another. On each execution of the inner loop.

f) We can use a similar approach to the one utilized in previous stages to accomplish this. With an ARRAY_SIZE = 64·1024 (4x cache size) and STEP_SIZE = 16·1024, we'll be accessing 4 elements of the array, all of which would be mapped to the same set in the cache. In a 4-way associative cache, all 4 elements would be able to fit into the set, so the AMAT should be about L1 access time. If the cache is a 2-way associative cache, only 2 elements would be able to be stored at once, which means that on every iteration of the inner loop we will have to access main memory, so AMAT will be Main memory access time.

4.5 e) **4 / 4**

✓ **- 0 pts** **Correct**

- 1 pts give more explanation

d) If the cache is DM, the AMAT should be equal to the main memory access time since, if the cache is DM, the 0th and $(16 \cdot 1024)^{th}$ element would map to the same block, leading to an eviction on each increment of the inner loop

e) If the cache is two-way set associative, the AMAT should be approximately the L1 access time since when both elements are mapped to the same set, there are 2 ways to store either element, so on each execution of the inner loop, the values don't need to be fetched from memory again since they aren't evicting one another. On each execution of the inner loop.

f) We can use a similar approach to the one utilized in previous stages to accomplish this. With an ARRAY_SIZE = 64·1024 (4x cache size) and STEP_SIZE = 16·1024, we'll be accessing 4 elements of the array, all of which would be mapped to the same set in the cache. In a 4-way associative cache, all 4 elements would be able to fit into the set, so the AMAT should be about L1 access time. If the cache is a 2-way associative cache, only 2 elements would be able to be stored at once, which means that on every iteration of the inner loop we will have to access main memory, so AMAT will be Main memory access time.

4.6 f) **5 / 5**

✓ **- 0 pts** Correct

(5) a)

A: 20 since we have 4KiB pages, so 1st 12 bits are for the page

B: 12 page size is 12 bits (2^{12}) for 4KiB

C: 24 physical address size is 36 bits so 36-12=24

D: 128 only have 128 entries

E: 26 Tag is all remaining bits in address: 36-5-6=25

F: 5 2^{11}=2KiB per way / 2^6 bytes per block = 2^5 blocks entries in set

G: 6 64 bytes per block w/ byte addressing, so 2^6=64 options to select byte

H: 32 2^5 entries in each set, use index bits to select entry

I: 20 Tag is all remaining bits, so 36-7-9=20

J: 9 $\frac{512\text{KiB}}{8\text{ ways}} / 128\text{B} = \frac{2^{19}/2^3}{2^7} = 2^9$ block entries in a set to choose from

K: 7 line size is 128, so $128 = 2^7$ ways to select which byte in line to output

L: 512 $2^9 = 512$ entries in each set of the cache

b) If the Intel cache is smaller than the AMD cache, it's hit time may be lower, which could account for the superior performance of the Intel cache. Alternatively, the AMD cache could have lower associativity (2-way rather than 4-way) which could account for the decreased performance in the AMD.

5.1 a) **6 / 6**

✓ **- 0 pts** **Correct**

- 1 pts some values are wrong

- 2 pts more than 5 values are wrong

(5) a)

A: 20 since we have 4KiB pages, so 1st 12 bits are for the page

B: 12 page size is 12 bits (2^{12}) for 4KiB

C: 24 physical address size is 36 bits so 36-12=24

D: 128 only have 128 entries

E: 26 Tag is all remaining bits in address: 36-5-6=25

F: 5 2^{11}=2KiB per way / 2^6 bytes per block = 2^5 blocks entries in set

G: 6 64 bytes per block w/ byte addressing, so 2^6=64 options to select byte

H: 32 2^5 entries in each set, use index bits to select entry

I: 20 Tag is all remaining bits, so 36-7-9=20

J: 9 $\frac{512\text{KiB}}{8\text{ ways}} / 128\text{B} = \frac{2^{19}/2^3}{2^7} = 2^9$ block entries in a set to choose from

K: 7 line size is 128, so $128 = 2^7$ ways to select which byte in line to output

L: 512 $2^9 = 512$ entries in each set of the cache

b) If the Intel cache is smaller than the AMD cache, it's hit time may be lower, which could account for the superior performance of the Intel cache. Alternatively, the AMD cache could have lower associativity (2-way rather than 4-way) which could account for the decreased performance in the AMD.

5.2 b) **4 / 4**

✓ **- 0 pts** Correct

- 0 pts see solution

c)

1) Mem lookup on TLB → TLB miss → Page Table Lookup
2) Page fault on Page Table → Mem transfer to PT
3) Loading Page from Physical memory causes eviction in PT
4) Mem Lookup restart → L1 cache miss
5) L2 lookup → L2 cache miss
6) Load data from DRAM → eviction in L2
7) Load data from L2 → eviction in L1

d) Using two sub blocks means that a 64 byte sub block can be fetched when needed rather than a full block. This is useful when we write to L2, but get a miss and need to fetch a block in order to write,

5.3 c) **6 / 6**

✓ **- 0 pts** Correct

c)

1) Mem lookup on TLB → TLB miss → Page Table Lookup
2) Page fault on Page Table → Mem transfer to PT
3) Loading Page from Physical memory causes eviction in PT
4) Mem Lookup restart → L1 cache miss
5) L2 lookup → L2 cache miss
6) Load data from DRAM → eviction in L2
7) Load data from L2 → eviction in L1

d) Using two sub blocks means that a 64 byte sub block can be fetched when needed rather than a full block. This is useful when we write to L2, but get a miss and need to fetch a block in order to write,

5.4 d) **2 / 2**

✓ **- 0 pts** **Correct**

- 1 pts provide proper reason

(6)

	Hit Time	Miss Rate	Miss Penalty
Double Associativity	**Increase** More ways to check on each lookup. i.e. Need to compare more tags to determine whether it's a hit or not	**Decrease** Decrease Conflict misses	**No Impact** Doesn't change higher level memory access time, or write time
Halving line Size	**Decrease** Will decrease the size of the cache, making lookups faster	**Increase** Will decrease capacity and decrease utilization of locality	**Decrease** Less information to transfer, less time required
Doubling # of sets	**Increase** Will decrease associativity since capacity const	**Decrease** Decrease conflict misses	**No Impact** Doesn't change higher level mem access time or write time
Adding Prefetching	**No impact** -Having the data requested more often does not impact the time needed to access the data, -Prefetching doesn't impact cache structure	**Decrease** Required data is available in the cache more often since Prefetching predicts use of data.	**Decrease ???** More data is transferred to the cache regularly, so there's a higher likelyhood requested data is available in a higher level cache, rather than main mem. if only an L1 cache, then **no impact**

6 6 **10 / 12**

- **0 pts** Correct
- **3 pts** 3 Cells are wrong
- **4 pts** 4 Cell are wrong
- **1 pts** 1 cell is wrong

✓ **- 2 pts** **2 cells are wrong**

- **10 pts** 10 cells are wrong