18-548/15-548
Memory System Architecture

Test #2

SOLUTIONS

Name (please print): _____

Instructions: ***DO NOT OPEN TEST UNTIL TOLD TO START***
YOU HAVE UNTIL 10:20 AM TO COMPLETE THIS TEST

The test is composed of three problems containing a total of 16 sub-problems, adding up to 100 points overall. The point value is indicated for each sub-problem. Attempt all problems and budget your time according to the problem's difficulty. Show all work in the space provided. If you have to make an assumption then state what it was. Answers unaccompanied by supporting work will not receive full credit. The exam is closed book, closed notes, and "closed neighbors." You are on your honor to have erased any course-relevant material from your calculator prior to the start of the test. Please print your initials at the top of each page in case the pages of your test get accidentally separated. You may separate the pages of the test if you like, and re-staple them when handing the test in.

Good luck!

1. Interleaved Memory & DRAM chip operation

You are building a customized data memory system for image processing, and need to access a single large matrix with an algorithm that reads all array elements by "row" (consecutive words) and subsequently reading them all by "column" (jumping by row size for each access). The following design information applies:

- 8192 x 8192 element array of 32-bit words (256 MB total array size)
- Use 64-Mbit DRAM chips organized as 16Mx4
  (identical in organization to the example DRAM chip presented in class)
- The DRAM chips take 24 bits of address, 11 of which are used for CAS and 13 of which are used for RAS. Thus, each DRAM chip can provide 2048 distinct 4-bit values when in page mode (RAS held low, multiple CAS cycles).
- Single-word RAS+CAS accesses to DRAM take: 60 ns of latency, 110 ns cycle time
- Page mode accesses to DRAM take: 25 ns of latency, 40 ns of cycle time.
  (Remember that a RAS-based access must come before any page mode/CAS-only accesses)
- The baseline design uses no interleaving. Byte addresses are converted to word addresses by dropping the bottom two bits. Of the resultant word address, the bottom 24 bits are used to select a DRAM location and the top address bits select the DRAM chip. Page mode access is supported for consecutive memory accesses to the same DRAM "page".
- The objective is to evaluate design alternatives to minimize the total time to perform both access patterns. Because of the size of the array and the algorithm, no cache memory will be used in the design – only DRAM chips.

a. (3 points)
What is the minimum number of DRAM chips necessary to contain this data array for the baseline design?

8Kx8Kx4 = 256 MB. Using 64 Mbit=8 Mbyte chips, this takes 32 chips.
The minimum number of chips to provide 32-bit memory words is 32/4=8, so that is not a driving constraint for this design

b. (8 Points)
What is the total time to read the array twice (once by row plus once by column) for the baseline design? (Show calculations for both by-row and by-column times)

By row accesses full pages of 2048 words/memory page = 8KB/page
256 MB / 8 KB = 32K pages.
One page takes 110 ns + 2047*40ns = 81,990 ns (no points off if 2048*40)
32K pages take 2.69 seconds

By column takes individual RAS cycles for each word = (256 MB / 4) * 110 ns = 7.38 seconds

Total time = 2.69 + 7.38 = 10.07 seconds just to touch memory

c. (7 points)
You decide to use five-way interleaved memory to hide the cycle time for single-word accesses (so, latency is still 60 ns, but you no longer have to wait the 110 ns before accessing the next word as long as it is to a different bank). You interleave based on low address bits, so each consecutive word is in the next bank. Given this change, show the time to access the array twice (once by row, once by column).

By row and by column now both take the same:

(256 MB / 4) * 60 ns = 4.027 sec
Total time = 4.027 * 2 = 8.05 seconds just to touch memory

d. (6 points)
Given that this is a special-purpose design, it is possible to provide even better memory bandwidth by organizing the 5-way interleaved memory to be interleaved on DRAM pages instead of on words. In other words, consecutive words in the same DRAM page are in the same module (bank), but once a DRAM page boundary is crossed the next word goes in the next memory module. Write the equation for the DRAM address (also called DRAM "word" address) and the equation module (bank) number for this approach. Be sure to state whether you are using byte or word addressing in your calculations.

DRAM word address = memory word address MOD 2048
or: = (memory byte address MOD 8192) / 4

DRAM module/bank number = (memory word address / 2048) MOD 5
or: = (memory byte address / 8192) MOD 5

e. (6 points)
For the design approach described in part d, what is the time required to read the array twice (once by row, once by column)?

By row is the same as in part b: 2.69 seconds

By column is the same as part c: 4.03 seconds

Total = 2.69 + 4.03 = 6.72 seconds – a lot better than with just interleaving.

2. Interleaved Memory & Chip area

You are doing a tradeoff for an on-chip unified L2 cache memory design. You have three design options which we will evaluate in turn in the sub-questions:
- Single-ported L2 cache (in a single memory array)
- Dual-ported L2 cache (in a single memory array)
- Two-way interleaved L2 cache (in a pair of memory arrays)

Design information is as follows:

- Total L2 cache size: 128 KB
- 15% of instructions accessing L2 cache require both instruction and data; the other 85% access either instruction or data in L2 (but not both).
  We are dealing with local miss ratio for L2 cache, and so these percentages only consider instructions that miss in L1 cache in the first place.
- Total size of each cache set: 163 bits, including 128 bits of cache data (one 16-byte block)
- Aspect ratio selected: Eight entire cache sets per memory row ("YMUX=8")
- Single-ported memory cell size = 1 micron on a side
  Dual-ported memory cell = 1.3 microns on a side (1.3 wide, 1.3 high)
- Width overhead for address decoder = 43 microns
  Height overhead for sense amps & multiplexors = 87 microns

a. (7 points)
What is the size of the single-ported L2 cache?

Width = 163 bits * 8 + 43 = 1347 microns
Height = 128 KB/(16*8) + 87 = 1111 microns size = 1347 * 1111 = 1,496,517 square microns

b. (3 points)
What is the effective access time for the single-ported L2 cache?
Access time for a single-ported memory array of this size is 5 ns. (Remember that instructions requiring both instruction and data from the L2 cache must cycle the cache twice.)

5 ns + 0.15 * 5 ns = 5.75 ns
(Fifteen percent of accesses require both I & D fetches, which means double-cycling the cache)

c. (7 points)
What is the size of the dual-ported L2 cache?

Width = 163 bits * 8 * 1.3 + 43 = 1738 microns
Height = (128 KB/(16*8)) * 1.3 + 87 = 1418 microns size = 1738 * 1418 = 2,464,484 square microns

(An even more correct solution might include area for a second address decoder and, possibly, a second sense amp/latch set.)

d. (3 points)
What is the effective access time for the dual-ported L2 cache?
Access time for a dual-ported memory array of this size is 7 ns.

7 ns (because instr+data can access concurrently)

(in real life, dual-ported memory access probably wouldn't be as slow as this compared to single-ported at the raw array level.)

e. (5 points)
What is the total size of the two-way interleaved L2 cache (sum of array sizes)?

Width = (163 bits * 8 + 43) = 1347 microns
Height = 64 KB/(16*8) + 87 = 599 microns size = 2 * (1347 * 599) = 1,613,706 square microns

f. (5 points)
What is the effective access time for the interleaved L2 cache?
Access time for an interleaved cache of this size is 5 ns. Assume that in 50% of accesses with both instruction and data there is a bank conflict.

5 ns + 0.15 * 0.5 * 5ns = 5.38 ns

(Note: This is a better speedup-per-area than the dual-ported memory.)

3. Software Tuning & Multi-Level Caching

You execute the following code fragment exactly one time on a machine with 32-bit int values.

```
#define OFFSET 1
int a[128][128], junka[OFFSET],
    b[128][128], junkb[OFFSET],
    c[128][128];
int i, j;

for (i = 0; i < 128; i++)
 { for (j = 0; j < 128; j++)
   { c[i][j] = b[i][j] + a[i][j];
   }
 }
```

The machine configuration is:

- 32-bit ints; 32-bit words; byte addressed on word boundaries
- 4 KB virtual memory page size
- L1 data cache:
  8 KB, 16-byte block size, 1 block/sector, direct mapped, *no-write-allocate*, *write-through*
  Hit time for L1 = 1 clock
- L2 data cache:
  256 KB, 16-byte block size, 1 block/sector, direct mapped, *write-allocate*, *write-back*
  Hit time for L2 = 1 clock to process L1 miss + 3 clocks to access L2 = 4 clocks
  Transport time for main memory (***after*** 4 clocks to discover there is an L2 miss) = 50 clocks
- Data TLB:
  128 entries available to program, 8 bytes per entry when placed in an inverted page table

Assume:

- Array a[] starts at address 0x00000000 and arrays are allocated contiguously. The assembly code ***always fetches b before a on a given iteration.***
- It just so happens that all physical addresses for data map identically to their virtual address (address translation is still performed, but for arrays a, b, and c their physical and virtual addresses just happen to be identical to simplify solving this problem)
- The machine starts with invalidated/"cold" caches and a "cold"/invalid data TLB
- Writing all modified data from the L2 cache to main memory is performed after the code runs, and is included in all answers to questions
- Only data cache behavior is considered (ignore instruction fetching)
- Caches are blocking, with no prefetching of data

a. (10 Points)
How many clocks are spent waiting for transport time for L2 cache misses? (In other words, this asks for number of clocks waiting for main memory beyond whatever clocks would have been required to access L2 cache if there had been hits.) Ignore TLB misses for this sub-problem.

Arrays are 128x128x4= 64 KB each. Thus three arrays will fit into L2 cache without wrap-around, so no conflicts
Given that this is a write-back cache, miss costs are as follows:

Array A is fetched for 128 x 128 words = 64 KB @ 16 bytes/ 50 clocks = 204,800 clocks
Array B is same as A = 204800 clocks + 50 clocks due to block mis-alignment = 204,850 clocks

Array C is write-allocate, and as such must be both read from memory (to fill missing words) and written back. Again, it's mis-aligned, so it is twice the Array B time = 409,700 clocks

Total time = 204,800 + 204,850 + 409,700 = 819,350 clocks.

b. (10 Points)
How many clocks are spent waiting for L1-to-L2 transport time (L1-to-L2 miss penalties)? (The L2 miss penalties are dealt with separately in part a). Ignore TLB misses for this sub-problem.

The 128x128 arrays wrap through an 8 KB cache exactly twice, with a 4-byte offset to the next array. Given that L1 is no-write-allocate, arrays a & b will be conflicting whenever elements share a block. Since arrays a & b are one word out of phase w.r.t. cache wraparound, this will lead to the following cases:

CASE	B block#	B offset	B hit/miss	A block#	A offset	A hit/miss
…	k	2	miss	k	1	miss
1	k	3	miss	k	2	miss
2	k+1	0	miss	k	3	hit
3	k+1	1	hit	k+1	0	miss
4	k+1	2	miss	k+1	1	miss
…	k+1	3	miss	k+1	2	miss

Thus there are 6 misses and 2 hits for every 4 iterations, but the very first iteration has no hit for B. Time for L2 access is thus 2 arrays * 128*128 * (6/8) * 3 clocks + 3 clocks = 73,731 clocks.

Additionally, there are 3 clocks for every access to array c = 128*128 * 3 = 49,152 clocks
Total = 73,731 + 49,142 = 122,873 clocks

c. (5 Points)
What is the minimum theoretical L1 data cache miss ratio for this program? Assume optimal placement of the arrays using the stated hardware design.

Minimum ratio is 1 miss of 4 for a & b + an alignment miss for b + 4 misses of 4 for c:
128*128 * 0.25 + 128*128 * 0.25 + 1 + 128*128 * 1.0 / (128*128*3) = 0.50
(this could be compared to the achieved miss ratio of 0.83 from part b, but this is not required to be stated by the question)

d. (5 Points)
What is the minimum non-negative value of OFFSET that could be used to reach the minimum theoretical L1 cache miss rate?

Almost correct: an offset of 4 would give a 16-byte separation between arrays a and b in cache, eliminating conflicts to blocks during program execution.

Completely correct: an offset of 3 suffices because b is accessed before a.

e. (10 Points)
How many clocks are spent processing TLB misses based on the data being fetched (no longer assume 0% TLB miss rate as had been stipulated in prior sub-problems)? Assume an inverted page table is already set up in main memory and the only data accesses required for TLB miss processing are to actually fetch an inverted page table entry. Note that this fetching is done via the data cache hierarchy

- TLB miss processing never causes any cache conflicts with data accesses; it takes 100 clocks plus memory hierarchy delay for fetching an 8-byte entry (two words) from the inverted page table
- Be sure to include time for L1 hits as well as all misses
- Assume "cold cache" with respect to inverted page table.

49 pages of virtual memory are touched while accessing data (the "junk" variables cause it to stretch into the 49th page). This requires 49 * 8 = 392 bytes of data to be loaded. Given the ground rules, they'll be in contiguous locations and stay in L1 cache as they are loaded. Total access time is therefore:
392/4 = 98 accesses to L1 cache * (1 clock memory + 100 clocks) = 9898 clocks
+ 98/4 rounded up = 25 L1 misses * 3 clocks = 75 clocks
+ 25 L2 misses * 50 clocks = 1250 clocks
= 11223 clocks.