18-742 Advanced Computer Architecture

Test I

February 24, 1998

SOLUTIONS

1. Cache Policies.

Consider a cache memory with the following characteristics:

- Unified, direct mapped
- Capacity of 256 bytes (0x100 bytes, where the prefix "0x" denotes a hexadecimal number)
- Two blocks per sector Each sector is 2 * 2 * 4 bytes = 0x10 bytes in size
- Two words per block
- 4-byte (32-bit) words, addressed in bytes but accessed only in 4-byte word boundaries
- Bus transfers one 32-bit word at a time, and transfers only those words actually required to be read/written given the design of the cache (i.e., it generates as little bus traffic as it can)
- No prefetching (i.e., demand fetches only; one block allocated on a miss)
- Write back policy; Write allocate policy

1a) (5 points) Assume the cache starts out completely invalidated. Circle one of four choices next to every access to indicate whether the access is a hit, or which type of miss it is. Put a couple words under each printed line giving a reason for your choice.

`read 0x0010 hit, compulsory miss, capacity miss, conflict miss`
First access is a miss

`read 0x0014 hit, compulsory miss, capacity miss, conflict miss`
Same block as 0x0010

`read 0x001C hit, compulsory miss, capacity miss, conflict miss`
Sector is already allocated, but this block in sector is still invalid

`read 0x0110 hit, compulsory miss, capacity miss, conflict miss`
First time access, evicts 0x0010, (credit given for "conflict miss" as an answer as well, but compulsory is a better answer if this is the very first time 0x0110 has been accessed)

`read 0x001C hit, compulsory miss, capacity miss, conflict miss`
Was just evicted by 0x0110

1b) (8 points) Assume the cache starts out completely invalidated. Circle "hit" or "miss" next to each access, and indicate the number of words of memory traffic generated by the access.

`write 0x0024 hit, miss, # words bus traffic = __1_____`
Compulsory miss, one word read in to fill block

`write 0x0020 hit, miss, # words bus traffic = __0_____`
Access to same block as before; no traffic

`write 0x0124 hit, miss, # words bus traffic = __3_____`
Compulsory miss; both words of block written out; one word read in to fill up the block

`read 0x0024 hit, miss, # words bus traffic = __4_____`
Conflict miss, both words written out & back in (only 1 dirty bit for the block)

2. Multi-level cache organization and performance.

Consider a system with a two-level cache having the following characteristics. The system does not use "shared" bits in its caches.

L1 cache	**L2 cache**
- Physically addressed, byte addressed	- Virtually addressed, byte addressed
- Split I/D	- Unified
- 8 KB combined, evenly split -- 4KB each	- 160 KB
- Direct mapped	- 5-way set associative; LRU replacement
- 2 blocks per sector	- 2 blocks per sector
- 1 word per block (8 bytes/word)	- 1 word per block (8 bytes/word)
- Write-through	- Write-back
- No write allocation	- Write allocate
- L1 hit time is 1 clock	- L2 hit time is 5 clocks (after L1 miss)
- L1 average local miss rate is 0.15	- L2 average local miss rate is 0.05

- The system has a 40-bit physical address space and a 52-bit virtual address space.
- L2 miss (transport time) takes 50 clock cycles
- The system uses a sequential forward access model (the "usual" one from class)

2a) (9 points)
Compute the following elements that would be necessary to determine how to configure the cache memory array:

Total number of bits within each L1 cache block: ____**65 bits**_____

8 bytes per block + 1 valid bit = 65 bits

Total number of bits within each L1 cache sector: ____**158 bits**_____

tag is 40 bits - 12 bits (4 KB) = 28 bits, and is the only sector overhead since it is direct mapped
28 + 65 + 65 = 158 bits

Total number of within each L2 set: _____**800 bits**_____

8 bytes per block + 1 valid bit + 1 dirty bit = 66 bits

cache is 32 KB * 5 in size ; tag is 40 bits - 15 bits (32 KB) = 25 bits tag per sector
+ 3 bits LRU counter per sector = 28 overhead bits per sector (one sector could also be only 37 bits per the trick discussed in class... but that is optional)

Set size = ((66 * 2) + 28) * 5) = 800 bits per set

2b) (5 points)
Compute the averate effective memory access time (t_{ea}, in clocks) for the given 2-level cache under the stated conditions.

$$t_{ea} = T_{hit1} + P_{miss1} T_{hit2} + P_{miss1} P_{miss2} T_{transport}$$

$$t_{ea} = 1 + (.15 * 5) + (.15 * .05 * 50) = \underline{\mathbf{2.125\ clocks}}$$

2c) (10 points)
Consider a case in which a task is interrupted, causing both caches described above to be completely flushed by other tasks, and then that original task begins execution again (i.e., a completely cold cache situation) and runs to completion, completely refilling the cache as it runs. What is the approximate time penalty, in clocks, associated with refilling the caches when the original program resumes execution? A restating of this same question is: assuming that the program runs to completion after it is restarted, how much longer (in clocks) will it take to run than if it had not been interrupted? For this sub-problem:

- The program makes only single-word memory accesses
- The reason we say "approximate" is that you should compute L1 and L2 penalties independently and then add them, rather than try to figure out coupling between them.
- State any assumptions you feel are necessary to solving this problem.

For the L1 cache, 1K memory references (8K / 8 bytes) will suffer an L1 miss before the L1 cache is warmed up. But, 15% of these would have been misses anyway. Extra L1 misses = 1K * 0.85 = 870 misses

L2 cache will suffer (160K / 8 bytes)* 0.95 extra misses = 19456 misses

Penalty due to misses is approximately:

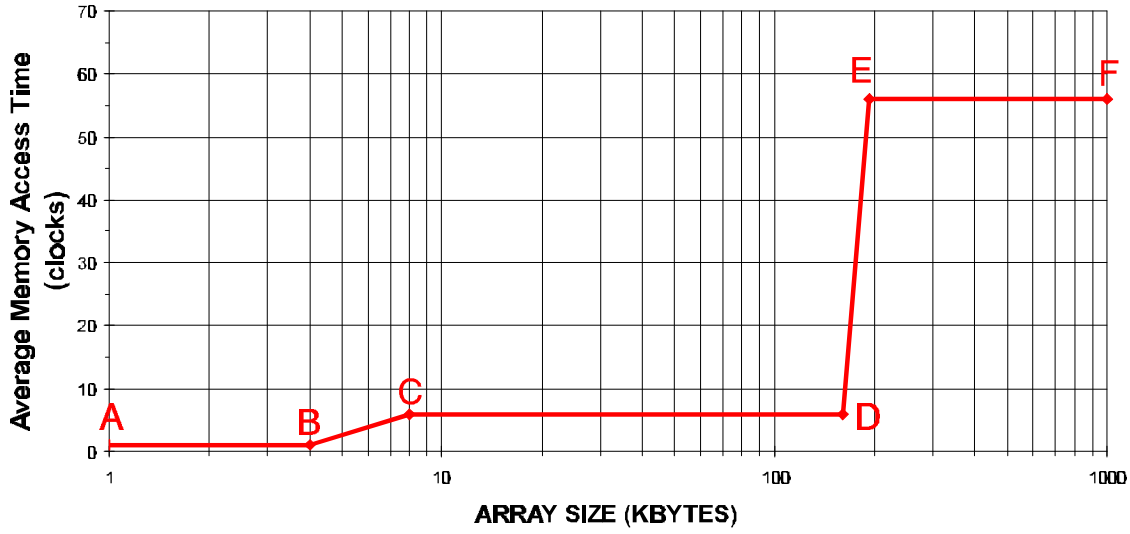
870 * 5 + 19456 * 50 = **<u>977,150 clocks</u> extra execution time due to loss of cache context**

A simpler answer but less accurate estimate that received 8 of 10 points was to ignore misses and just compute the raw fill times of the cache, giving 1,029,120 clocks (assumes 100 miss rate until the caches are full)

2d) (10 points)
For the given 2 level cache design, sketch the *data cache* behavior of the following program in the graphing area provided (ignore all instruction accesses -- consider data accesses only). Label each point of inflection (i.e., points at which the graph substantially changes behavior) and the end points on the curve with a letter and put values for that point in the data table provided along with a brief statement (3-10 words) of what is causing the change in behavior and how you knew what array size it would change at. The local miss rates given earlier are for a "typical" program, and obviously do not apply to the program under consideration in this sub-question.

```
int touch_array(long *array, int size)
{ int i;
  int sum=0;
  long *p, *limit;
  limit = array + size/sizeof(int);
  for (i = 0; i < 1000000; i++)
  { /* sequentially touch all elements in array */
    for (p = array; p != limit; p++)
    { sum += *p; }
  }
  return(sum);
}
```

Point	Array size	Data t_{ea}	Reason
A	1 KB	1	All L1 Hits
B	4 KB	1	All L1 Hits (L1 D-cache 4 KB)
C	8 KB	6	L1 miss; L2 hit (twice L1 size)
D	160 KB	6	L1 miss; L2 hit (L2 size)
E	192 KB	56	L2 miss; L2 * (1 + 1/5) size
F	1000 KB	56	All L2 Misses



3. Virtual Memory.

Consider a virtual memory system with the following characteristics:

- Page size of 16 KB = 2^{14} bytes
- Page table and page directory entries of 8 bytes per entry
- Inverted page table entries of 16 bytes per entry
- 31 bit physical address, byte addressed
- Two-level page table structure (containing a page directory and one layer of page tables)

3a) (9 points) What is size (in number of address bits) of the virtual address space supported by the above virtual memory configuration?

Page size of 16 KB gives 2^{14} address space = 14 bits

Each Page Table has 16 KB/8 bytes of entries = $2^{14}/2^3$ = 11 bits of address space

The Page Directory is the same size as a page table, and so contributes another 11 bits of address space

14 + 11 + 11 = 36 bit virtual address space.

3b) (10 points) What is the maximum required size (in KB, MB, or GB) of an inverted page table for the above virtual memory configuration?

There must be one inverted page table entry for each page of physical memory.

$2^{31}/2^{14}$ = 2^{17} pages of physical memory, meaning the inverted page table must be

2^{17} * 16 bytes = 2^{17} * 2^4 bytes = 2^{21} = 2 MB in size

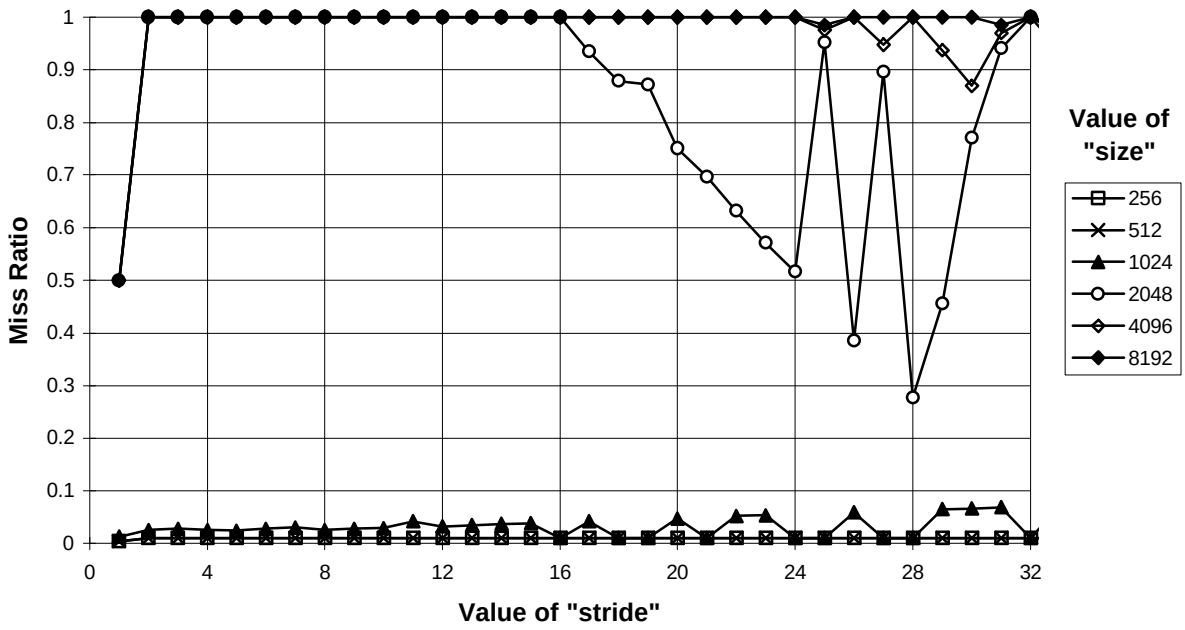
4. Cache simulation

The following program was instrumented with Atom and simulated with a cache simulator. The cache simulator was fed only the data accesses (not instruction accesses) for the following subroutine:

```
long test_routine(long *a, int size, int stride)
{ long result=0;
  long *b, *limit;
  int i;

  limit = a + size;
  for (i = 0; i < 100; i++)
  { b = a;
    while (b < limit)
    { result += *b;
      b += stride;
    }
  }
  return(result);
}
```

The result of sending the Atom outputs through the cache simulator for various values of "size" and "stride" (each for a single execution of `test_routine`) are depicted in the graph below. The cache word size was 8 bytes (matching the size of a "long" on an Alpha), and a direct-mapped cache was simulated. The simulated cache was invalidated just before entering `test_routine`. There was no explicit prefetching used. The code was compiled with an optimization switch that got rid of essentially all reads and writes for loop handling overhead.
Use only "Cragon" terminology for this problem (not dinero terminology).



4a) (9 points)
For the given program and given results graph, what was the size of the cache in KB (support your answer with a brief reason)? Remember that in the C programming language pointer arithmetic is automatically scaled according to the size of the data value being pointed to (so, in this code, adding 1 to a pointer actually adds the value 8 to the address).

Cache size = 2048 * 8 / 2 = **8 KB**

Direct mapped caches have high 100% miss rate when array size >= 2* cache size

4b) (10 points)
For this example, what is the cache block size in bytes (support your answer with a brief reason)?

Cache block size is 2 words = **16 bytes**, because with a stride of 1 the large arrays have 50% miss rate, indicating that odd-word-addressed words are fetched as a pair with even-word-addressed cache misses, yielding spatial locality cache hits when both even and odd words in the same block are touched.

4c) (15 points)
What parameter is responsible for the "size"=2K (the data plotted with circles) point of inflection at a "stride" of 16? What is the value of that parameter, and why?

Cache **sector size** is 16 * 8 = **128 bytes**; because above this value sectors are skipped on the first pass through the cache but filled on the second pass given that array size is twice cache size. Alternately, this could be described as having 8 blocks per sector.

For a stride of 17, every 16th access skips an entire sector, leaving it still resident in cache for the next iteration through the loop for an 8K array access. (When the strided accesses wrap around, the second iteration through the cache will touch those locations and remain in cache.) The effect is less pronounced for larger arrays, but there nonetheless. The 100% miss rate at 32 (sector size * 2) confirms that factors of 16 result in maximal conflict+capacity misses. The graph dips further as stride increases because more sectors are skipped the in the first half of the array, leaving more "holes" for data as the second half of the array maps into the cache.

An alternate answer is that the parameter is the number of sets=sectors in the cache.