18-742 Advanced Computer Architecture

Test I

February 24, 1998

Name (please print): _____

Instructions: ***DO NOT OPEN TEST UNTIL TOLD TO START***
YOU HAVE UNTIL 12:20 PM TO COMPLETE THIS TEST

The exam is composed of four problems containing a total of 11 sub-problems, adding up to 100 points overall. The point value is indicated for each sub-problem and is roughly proportional to difficulty. 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!

Grading Sheet

1a) (5) _____

1b) (8) _____

2a) (9) _____

2b) (5) _____

2c) (10) _____

2d) (10) _____

3a) (9) _____

3b) (10) _____

4a) (9) _____

4b) (10) _____

4c) (15) _____

TOTAL: (100) _____

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
- 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

read 0x0014 hit, compulsory miss, capacity miss, conflict miss

read 0x001C hit, compulsory miss, capacity miss, conflict miss

read 0x0110 hit, compulsory miss, capacity miss, conflict miss

read 0x001C hit, compulsory miss, capacity miss, conflict miss

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 = _____

write 0x0020 hit, miss, # words bus traffic = _____

write 0x0124 hit, miss, # words bus traffic = _____

read 0x0024 hit, miss, # words bus traffic = _____

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	- Physically 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: _____

Total number of bits within each L1 cache sector: _____

Total number of bits within each L2 set: _____

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

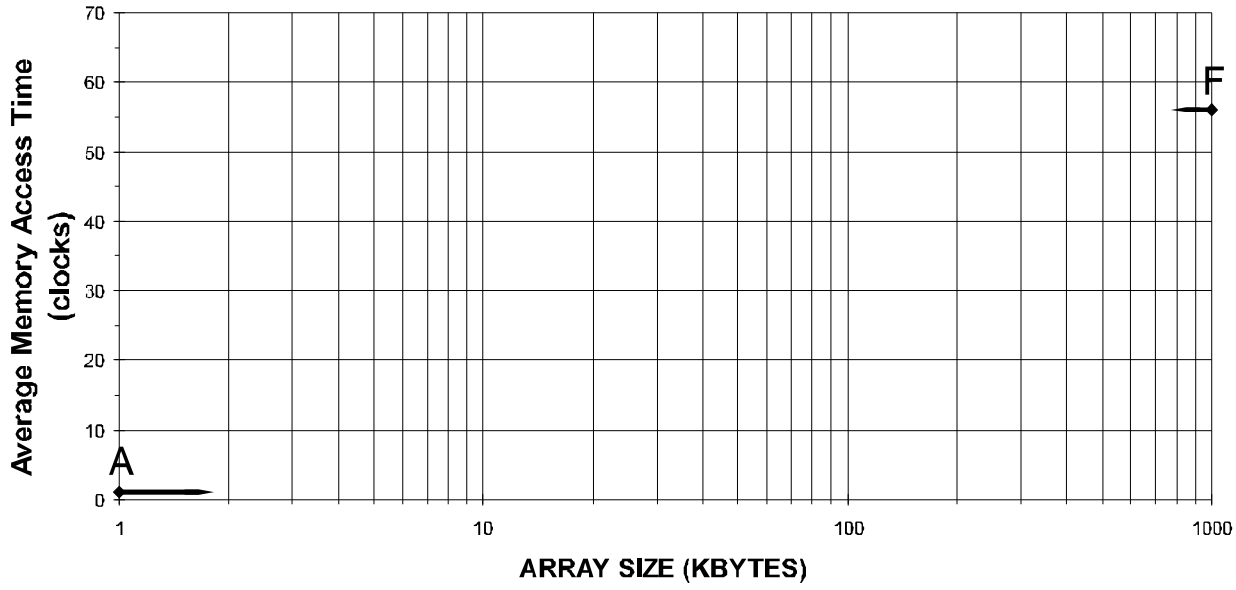
2c) (10 points)
Consider a case in which a task is interrupted, causing both caches described above to be completely flushed by other tasks, and then that original task begins execution again (i.e., a completely cold cache situation) and runs to completion, completely refilling the cache as it runs. 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.

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.

```
long touch_array(long *array, int size)
{ int i;
  long 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	t_{ea}	Reason
A	1 KB	1	All L1 Hits
B			
C			
D			
E			
F	1000 KB		All L2 Misses



3. Virtual Memory.

Consider a virtual memory system with the following characteristics:

- Page size of 16 KB
- 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?

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?

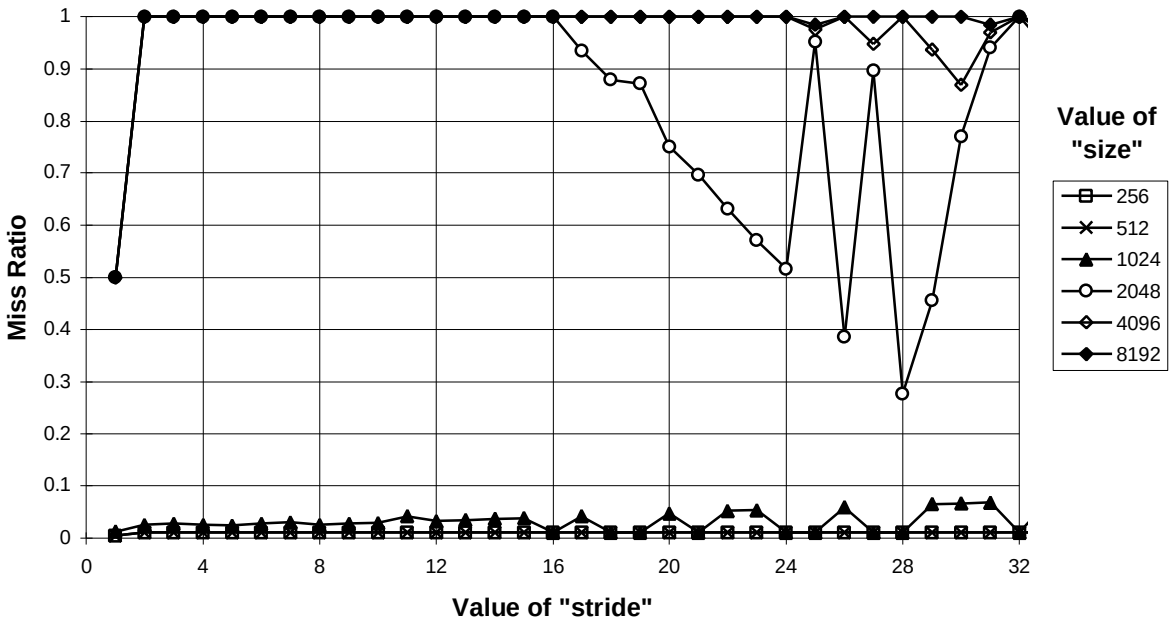
4. Cache simulation

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).

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

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?