18-548/15-548
Memory System Architecture

Test #1

SOLUTIONS

September 28, 1998

1. Cache Policies & Operation

Consider a uniprocessor data cache with the following design parameters:

- 8 Kbytes data capacity = 0x2000 bytes
- Byte addressed on 64-bit word boundaries = 8 bytes/word
- 4 words per block = 4 words/block = 0x0020 bytes/block
- 2 blocks per sector = 8 words/sector = 0x0040 bytes
- 2-way set associative; LRU replacement = 0x1000 bytes before "wrap-around"
- Write allocate; write-back

Assume cache starts out completely invalidated. For each memory reference of the reference stream below, annotate the reference with the number of words transferred between cache and memory as a result of that reference. Circle the type of miss/hit it is (one of the following: "hit", "compulsory miss", "capacity miss", "conflict miss"), and provide a few words of rationale in the space under each access. Addresses given are hexadecimal byte addresses. Assume that no extraneous (avoidable) words are transferred to or from cache.

(5 points/address below x 5 addresses = 25 points):

Operation **# Words** **Hit/Miss type**

1a) read 0x1110__4____ hit / compulsory / capacity / conflict

Allocates 0x1100 in first set; reads 4 words in the block

1b) write 0x0100__3____ hit / compulsory / capacity / conflict

Allocates 0x0100 second set; reads 0x0108..0x0118 to fill block

1c) read 0x1138__4____ hit / compulsory / capacity / conflict

Fills second block at 0x1100 and updates LRU field

1d) write 0x5130__7____ hit / compulsory / capacity / conflict

Evicts 0x0100 sector (writing 4 words); allocates & reads 3 words to fill block

1e) read 0x5120__0____ hit / compulsory / capacity / conflict

Hit on data filled on the write to 5130.

2. Latency/Bandwidth tradeoff

Consider a sequential forward cache design (the "usual" design) on an embedded processor. The system you are building uses 64-bit words for floating point control computations. You have determined the system will have the following characteristics:

- 300 MHz CPU; 50 MHz bus speed (6 CPU clocks = 1 bus clock)
- Cache has two 64-bit words per block and one block per sector
- Bus uses burst transfer mode: 4 bus clocks for first datum, and 1 bus clock per datum in burst
- Bus width is 16 data bits (16 bits transferred per bus clock during a burst)
- Cache hits take one CPU clock
- 6% data cache miss ratio

2a) (10 points)

Considering only data accesses (ignoring instruction accesses), what is the effective memory access time t_{ea} for this system in *nanoseconds*?

$$t_{ea} = hit\_time + P_{miss} * transport\_time$$

$$t_{ea} = hit\_time + P_{miss} * \left[a + b\left(\frac{A}{W} - 1 \right) \right]$$

$$t_{ea} = 1 + .06 * \left[24 + 6\left(\frac{2}{.25} - 1 \right) \right] = 4.96 \text{ CPU clocks} = 16.5 \text{ ns}$$

Alternate solution:

The transfer is 128 / 16 bits = 8 transfers on the bus; 4 clocks + 7 = 11 bus clocks/transfer

1 CPU clock + 0.06 * (11 * 6) = 4.96 CPU clocks = 16.5 ns at 300 MHz

2b) (15 points)
Which speedup would be ***greater*** between the following two options if nothing else were changed:

OPTION I) doubling bus width to 32 bits (at the same clock frequencies, but requires fewer bus clocks in a burst transfer)
OR
OPTION II) increasing the bus speed and memory access time to support a 100 MHz bus clock speed. This means the number of bus clocks for a cache miss would remain constant, but be done twice as fast

There are several ways to reach a correct answer; two of them are below. Note that minimal points were subtracted for errors that were the same in (2a) as (2b).

$$t_{ea}\_32bitbus = 1 + .06 * \left[24 + 6\left(\frac{2}{.5} - 1 \right) \right] = 3.52 \text{ CPU clocks} = 11.73 \text{ ns}$$

$$t_{ea}\_100Mhz = 1 + .06 * \left[12 + 3\left(\frac{2}{.25} - 1 \right) \right] = 2.98 \text{ CPU clocks} = 9.93 \text{ ns}$$

tea approach:
The 100 MHz approach is much faster, because it speeds up the initial latency as well as the burst transfer. Note that you can tell which is greater by tea without actually calculating speedup.

For the Amdahl's Law approach you have to compute the fraction of time in the original system spent on cache misses = .06 * (67) / 4.96 = 0.81

$$speedup = \frac{1}{(1 - fraction\_enhanced) + \left(\frac{Fraction\_enhanced}{Speedup\_enhanced} \right)}$$

$$speedup\_32 = \frac{1}{(1 - .81) + \left(\frac{.81}{\left(\frac{67}{43} \right)} \right)} = 1.41$$

$$speedup\_100MHz = \frac{1}{(1 - .81) + \left(\frac{.81}{\left(\frac{67}{1 + 66/2} \right)} \right)} = 1.66 \quad \text{this is a bigger speedup}$$

The 100 MHz approach is much faster, because it speeds up the initial latency as well as the burst transfer.

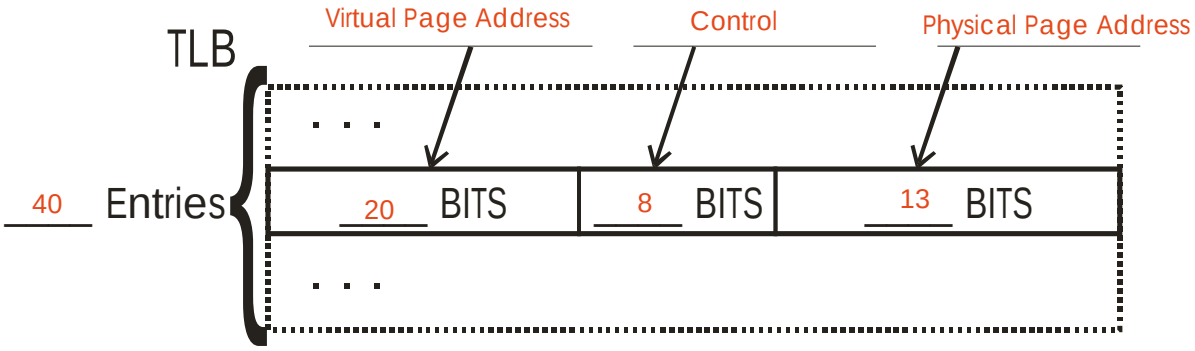
---

3. Virtual Memory

The Epsilon is a uniprocessor with the following memory configuration:

Physical Memory	**Virtual Memory**	**TLB**
physical address space is 64 MB	virtual page size is 8 KB	unified
byte-addressable	virtual address space is 8 GB	fully associative
transfers in 4 byte words	two-level page table	LRU replacement
	8 byte page table entries	40 entries
	8 byte page directory entries	1 byte control/entry

a) (10 points) Label the following diagram of the TLB, including the width in bits of each section of the representative TLB entry drawn below. Note that the picture may not be to scale. Full credit requires the three boxes to be labeled, three bit widths to be specified in the blanks provided, and the number of entries to be stated in the blank provided. Show your work for computing the bit widths.



Control is 8 bits per the problem statement
40 entries per the problem statement

virtual page address is 8 GB / 8KB page size = 2**20, giving 20 bits

physical page address is 64 MB / 8 KB page size = 8 MB = 2**13, giving 13 bits

Additionally, for sub-problems 3b and 3c:

- consider a representative program which reads a single array
- this array has been allocated starting at address zero
- all data accesses of programs are in 4-byte words
- there is no inverted page table

b) (10 points) A hapless '548 student decides to stress the memory system by reading 2 KB of data in 4-byte words, but doing it with a stride of 8 MB (so, each 4-byte word is at an address 8 MB higher than the previous word, and there are 512 such accesses). How many **total bytes** (please be exact) of main memory will be allocated to data for that task assuming only pages that are touched are allocated (include "overhead" pages touched due to the virtual memory organization described).

Iterating in 8 MB strides is nasty because it forces the allocation of one entire page table for every data page read.
512 pages for the data
+ 512 pages for the page table (at one per data touch)
+ 1 page for the page directory
= 1025 * 8K = 8,396,800 bytes allocated just to read 2 KB of data --- Phew!

c) (5 points) If a system designer decided to add an inverted page table to this system, how big must it be at a minimum? (Make reasonable assumptions as required and state them.)

Physical address space is 64 MB, assume a fully populated main memory. This is in 8 KB pages, giving a minimum of 64 MB/8KB = 8K entries. Assume Inverted Page Table entries are the same size as TLB entries at 8 bytes. 8K * 8 bytes = 64 KB inverted page table size.

4. Cache performance interpretation

Consider the below program, which is nearly identical to what you saw in lab #2 (except the variables are all 64-bit "long" instead of 32-bit "int"). The program is timed for successively larger values of `size`, and this process is repeated for successively larger values of `stride`. Note that `size` and `stride` are both in terms of 8-byte long values, not bytes.

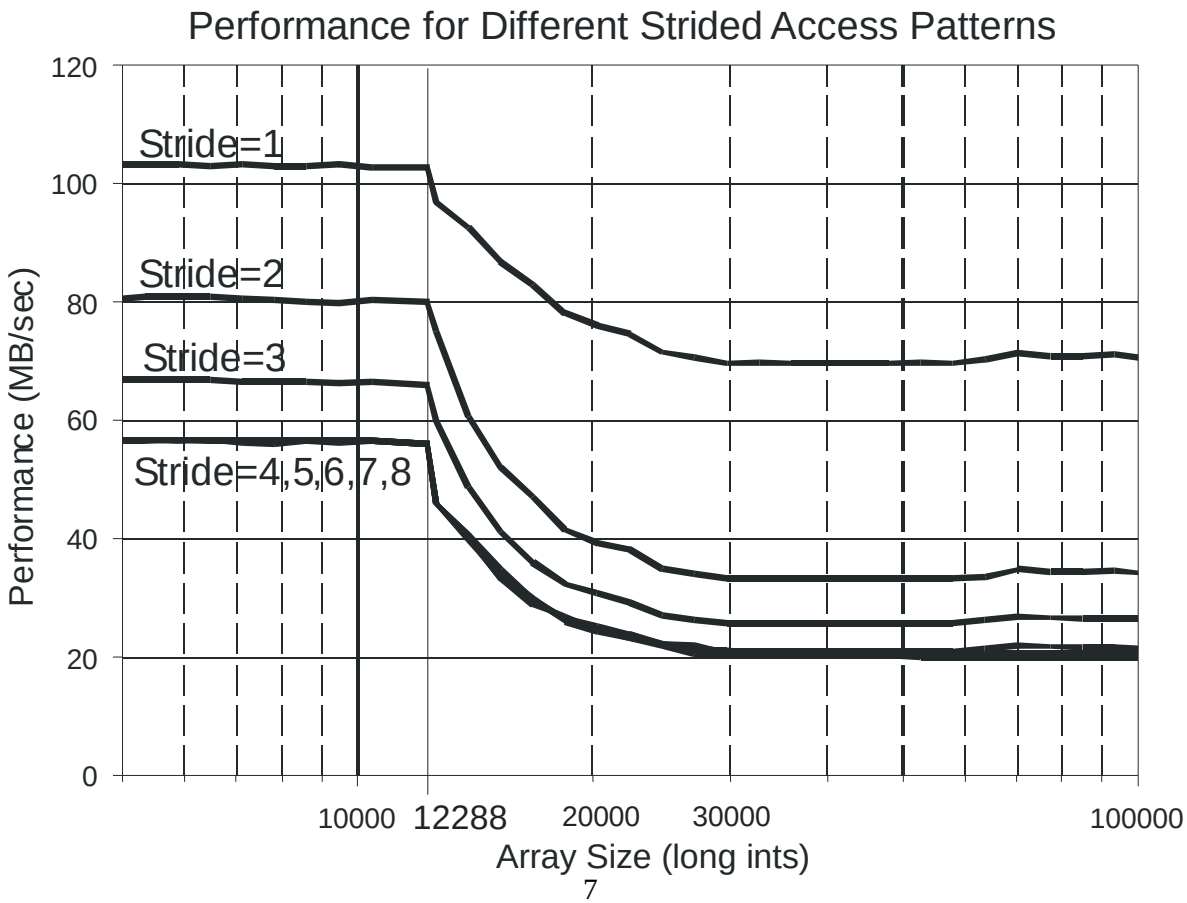
```
int touch_array(long *array, long size, long stride, long iter)
{ long i;
  long sum=0;
  long *p, *limit;

  limit = &(array[size-1]);

  for (i = 0; i < iter; i++)
  { /* touch elements; pointer match results in faster code */
    for (p = array; p < limit; p+= stride)
    { sum += *p; }
  }
  return(sum);
}
```

The data from running this program on a single level of cache is as below. The Y axis is performance in MB/sec; the X axis is array size in "longs" (8-bytes per "long"); the curves correspond to different values of `stride` in longs (e.g., stride=3 means a 24-bytes stride).

The data chart given is real performance data from an Alphaserver 4000.



4a) (8 Points)
How big is this cache and how did you determine that?

The performance drops at stride=12K, which is a size of 96KB for cache.

4b) (10 Points)
What is the block size of this cache and how did you determine that?

Stride values of 1, 2, and 3 have better performance than strides of 4 or more, meaning that they have some fraction of accesses that hit multiple times in the same block. Therefore block size is stride of 4 = 8*4 = 32 bytes.

4c) (7 Points)
The cache is 3-way set associative. Is the replacement policy LRU or Random (and, more importantly, why?)

If LRU, you would expect the performance drop to go from stride=12K to stride=16K, which is a drop over 1/3rd of the cache size (for 3-way set associative). It goes further, to almost 30K, so it must be random (note that after about 20K it gets pretty flat – about what you'd expect for random).