6.004 Computation Structures
Spring 2009

Memory hierarchy

Problem 1. The following is a sequence of address references given as word addresses:

2,3,11,16,21,13,64,48,19,11,3,22,4,27,6,11

A. ★ Show the hits and misses and final cache contents for a fully associative cache with one-word blocks and a total size of 16 words. Assume LRU replacement.

2: miss, cache now holds: 2
3: miss, cache now holds: 3, 2
11: miss, cache now holds: 11, 3, 2
16: miss, cache now holds: 16, 11, 3, 2
21: miss, cache now holds: 21, 16, 11, 3, 2
13: miss, cache now holds: 13, 21, 16, 11, 3, 2
64: miss, cache now holds: 64, 13, 21, 16, 11, 3, 2
48: miss, cache now holds: 48, 64, 13, 21, 16, 11, 3, 2
19: miss, cache now holds: 19, 48, 64, 13, 21, 16, 11, 3, 2
11: hit, cache now holds: 11, 19, 48, 64, 13, 21, 16, 3, 2
3: hit, cache now holds: 3, 11, 19, 48, 64, 13, 21, 16, 2
22: miss, cache now holds: 22, 3, 11, 19, 48, 64, 13, 21, 16, 2
4: miss, cache now holds: 4, 22, 3, 11, 19, 48, 64, 13, 21, 16, 2
27: miss, cache now holds: 27, 4, 22, 3, 11, 19, 48, 64, 13, 21, 16, 2
6: miss, cache now holds: 6, 27, 4, 22, 3, 11, 19, 48, 64, 13, 21, 16, 2
11: hit, cache now holds: 11, 6, 27, 4, 22, 3, 19, 48, 64, 13, 21, 16, 2

B. ★ Show the hits and misses and final cache contents for a fully associative cache with *four*-word blocks and a total size of 16 words. Assume LRU replacement.

With a N-word block of data for each cache entry, note that the N words in a cache entry will have consecutive memory addresses *starting with a word address that's a multiple of N*.

2: miss, cache now holds: 0-3
3: hit, cache now holds: 0-3
11: miss, cache now holds: 8-11, 0-3
16: miss, cache now holds: 16-19, 8-11, 0-3
21: miss, cache now holds: 20-23, 16-19, 8-11, 0-3
13: miss, cache now holds: 12-15, 20-23, 16-19, 8-11

64: miss, cache now holds: 64-67, 12-15, 20-23, 16-19
48: miss, cache now holds: 48-51, 64-67, 12-15, 20-23
19: miss, cache now holds: 16-19, 48-51, 64-67, 12-15
11: miss, cache now holds: 8-11, 16-19, 48-51, 64-67
3: miss, cache now holds: 0-3, 8-11, 16-19, 48-51
22: miss, cache now holds: 20-23, 0-3, 8-11, 16-19
4: miss, cache now holds: 4-7, 20-23, 0-3, 8-11
27: miss, cache now holds: 24-27, 4-7, 20-23, 0-3
6: hit, cache now holds: 4-7, 24-27, 20-23, 0-3
11: miss, cache now holds: 8-11, 4-7, 24-27, 20-23

Problem 2. Cache multiple choice:

A. ★ If a cache access requires one clock cycle and handling cache misses stalls the processor for an additional five cycles, which of the following cache hit rates comes closest to achieving an average memory access of 2 cycles?

(A) 75%
(B) 80%
(C) 83%
(D) 86%
(E) 98%

2 cycle average access = (1 cycle for cache) + (1 - hit rate)(5 cycles stall)
=> hit rate = 80%

B. ★ LRU is an effective cache replacement strategy primarily because programs

(A) exhibit locality of reference
(B) usually have small working sets
(C) read data much more frequently than write data

(A). Locality implies that the probability of accessing a location decreases as the time since the last access increases. By choosing to replace locations that haven't been used for the longest time, the least-recently-used replacement strategy should, in theory, be replacing locations that have the lowest probability of being accessed in the future.

C. ★ If increasing the block size of a cache improves performance it is primarily because programs

(A) exhibit locality of reference

(B) usually have small working sets
(C) read data much more frequently than write data

(A). Increased block size means that more words are fetched when filling a cache line after a miss on a particular location. If this leads to increased performance, then the nearby words in the block must have been accessed by the program later on, ie, the program is exhibiting locality.

D. ★ Consider the following program:

```
integer A[1000];
for i = 1 to 1000
  for j = 1 to 1000
    A[i] = A[i] + 1
```

When the above program is compiled with all compiler optimizations turned off and run on a processor with a 1K byte fully-associative write-back data cache with 4-word cache blocks, what is the approximate data cache miss rate? (Assume integers are one word long and a word is 4 bytes.)

(A) 0.0125%
(B) 0.05%
(C) 0.1%
(D) 5%
(E) 12.5%

(A). Considering only the data accesses, the program performs 1,000,000 reads and 1,000,000 writes. Since the cache has 4-word blocks, each miss brings 4 words of the array into the cache. So accesses to the next 3 array locations won't cause a miss. Since the cache is write-back, writes happen directly into the cache without causing any memory accesses until the word is replaced. So altogether there are 250 misses (caused by a read of A[0], A[4], A[8], ...), for a miss rate of 250/2,000,000 = 0.0125%

E. ★ In a non-pipelined single-cycle-per-instruction processor with an instruction cache, the average instruction cache miss rate is 5%. It takes 8 clock cycles to fetch a cache line from the main memory. Disregarding data cache misses, what is the approximate average CPI (cycles per instruction)?

(A) 0.45
(B) 0.714
(C) 1.4
(D) 1.8
(E) 2.22

(C). CPI = (1 inst-per-cycle) + (0.05)(8 cycles/miss) = 1.4

Problem 3. A student has miswired the address lines going to the memory of an unpipelined BETA. The wires in question carry a 30-bit word address to the memory subsystem, and the hapless student has in fact reversed the order of all 30 address bits. Much to his surprise, the machine continues to work perfectly.

A. Explain why the miswiring doesn't affect the operation of the machine.

Since the Beta reverses the order of the 30 bit address in the same manner for each memory access, the Beta will use the same reversed address to access a particular memory location for both stores and loads. Thus, the operation of the machine will not be affected.

B. The student now replaces the memory in his miswired BETA with a supposedly higher performance unit that contains both a fast fully associative cache and the same memory as before. The reversed wiring still exists between the BETA and this new unit. To his surprise, the new unit does not significantly improve the performance of his machine. In desperation, the student then fixes the reversal of his address lines and the machine's performance improves tremendously. Explain why this happens.

Caches take advantage of locality of reference by reading in an entire block of related data at one time, thereby reducing main memory accesses. By reversing the order of the 30 bit address, locality of the memory addresses is disrupted. The low-order bits that would normally place related data close to one another are instead the high-order bits and related data is more spread out through the main memory. This reduction in locality reduces cache performance significantly. When the student fixes the address line reversal problem, locality of the memory is restored, and the cache can perform as intended.

Problem 4. For this problem, assume that you have a processor with a cache connected to main memory via a bus. A successful cache access by the processor (a hit) takes 1 cycle. After an unsuccessful cache access (a miss), an entire cache block must be fetched from main memory over the bus. The fetch is not initiated until the cycle following the miss. A bus transaction consists of one cycle to send the address to memory, four cycles of idle time for main-memory access, and then one cycle to transfer each word in the block from main memory to the cache. Assume that the processor continues execution only after the last word of the block has arrived. In other words, if the block size is B words (at 32 bits/word), a cache miss will cost 1 + 1 + 4 + B cycles. The following table gives the average cache miss rates of a 1 Mbyte cache for various block sizes:

Block size (B)	*Miss ratio (m), %*
1	3.4
4	1.1
8	0.43
16	0.28
32	0.19

A. ★ Write an expression for the average memory access time for a 1-Mbyte cache and a B-word block size (in terms of the miss ratio m and B).

Average access time = (1-m)(1 cycle) + (m)(6 + B cycles) = 1 + (m)(5+B) cycles

B. ★ What block size yields the best average memory access time?

Block size B (m)	*Access time Part (B) 1+m(5+B)*
1 (.034)	1.204
4 (.011)	1.099
8 (.0043)	*1.056*
16 (.0028)	1.059
32 (.0019)	1.0703

C. If bus contention adds three cycles to the main-memory access time, which block size yields the best average memory access time?

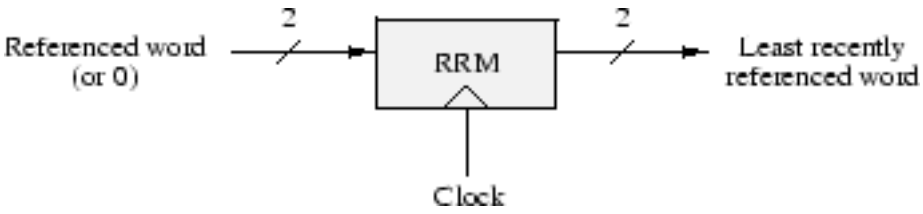
Block size B (m)	*Access time Part (C) 1+m(8+B)*
1 (.0034)	1.306
4 (.0011)	1.132
8 (.0043)	1.069
16 (.0028)	*1.067*
32 (.0019)	1.076

D. If bus width is quadrupled to 128 bits, reducing the time spent in the transfer portion of a bus

transaction to 25% of its previous value, what is the optimal block size? Assume that a minimum one transfer cycle is needed and don't include the contention cycles introduced in part (C).

Block size B (m)	*Access time Part (D) 1+m(5+roundup[B/4])*
1 (.0034)	1.204
4 (.0011)	1.066
8 (.0043)	1.0301
16 (.0028)	1.0252
32 (.0019)	*1.0247*

Problem 5. You are designing a controller for a tiny cache that is fully associative but has only three words in it. The cache has an LRU replacement policy. A reference record module (RRM) monitors references to the cache and always outputs the binary value 1, 2, or 3 on two output signals to indicate the least recently used cache entry. The RRM has two signal inputs, which can encode the number 0 (meaning no cache reference is occurring) or 1, 2, or 3 (indicating a reference to the corresponding word in the cache).



A. What hit ratio will this cache achieve if faced with a repeating string of references to the following addresses: 100, 200, 104, 204, 200?

Here's what happens:

```
access 100: miss; cache contains 100, ---, ---
access 200: miss; cache contains 200, 100, ---
access 104: miss; cache contains 104, 200, 100
access 204: miss; cache contains 204, 104, 200
access 200: hit;  cache contains 200, 204, 104
access 100: miss; cache contains 100, 200, 204
access 200: hit;  cache contains 200, 100, 204
access 104: miss; cache contains 104, 200, 100
access 204: miss; cache contains 204, 104, 200
```

```
access 200: hit;  cache contains 200, 204, 104
...
```

So in the steady state, location 200 stays in the cache and all other locations get replaced. So the hit rate is 2/5 or 40%.

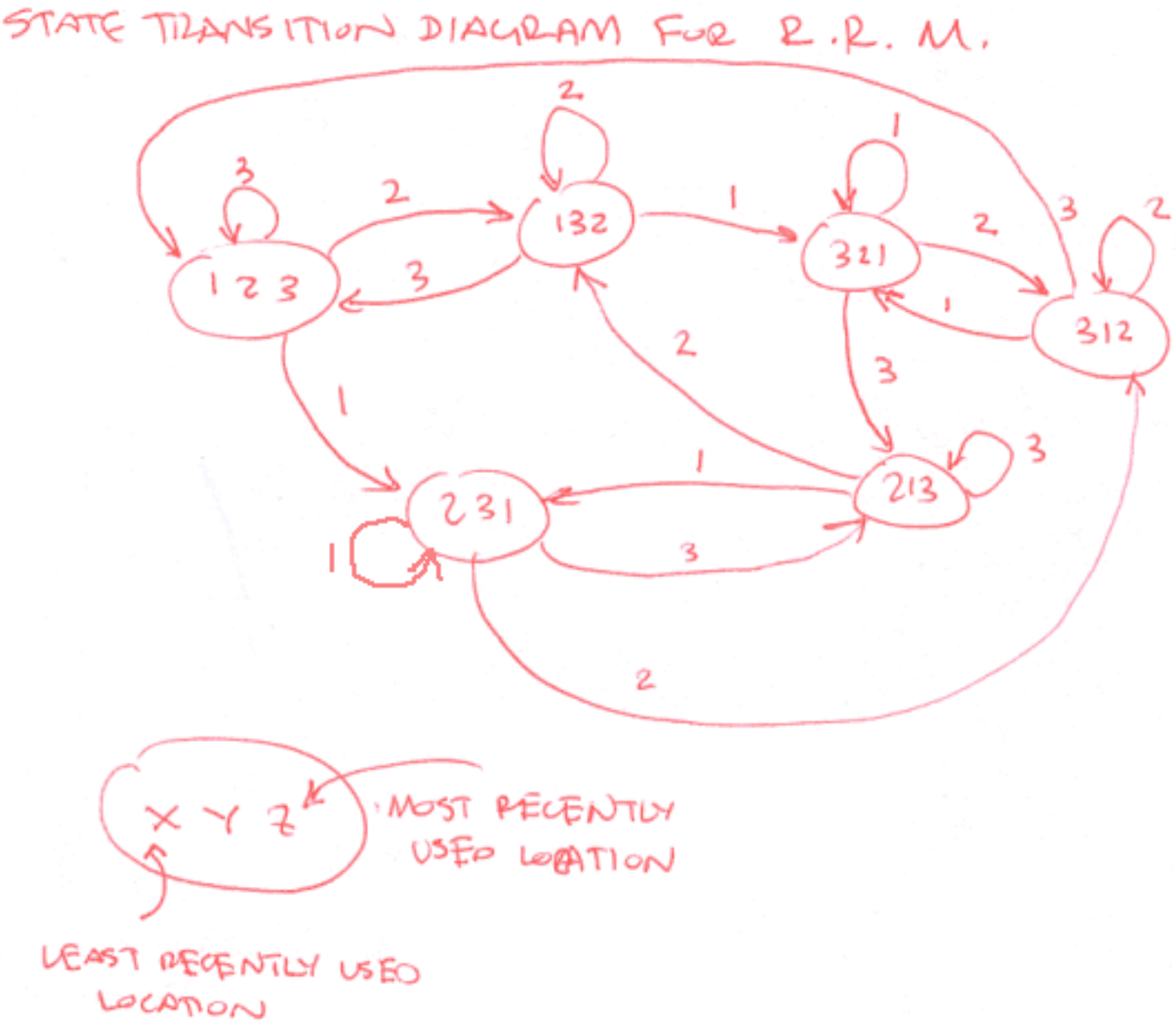
B. The RRM can be implemented as a finite-state machine. How many states does the RRM need to have? Why?

There are 3! = 6 ways to list the three locations in order of use. Thus RRM needs 6 states, one state for each possible order.

C. How many state bits does the RRM need to have?

We can encode six states using 3 state bits.

D. Draw a state-transition diagram for the RRM.



E. Consider building an RRM for a 15-word fully associative cache. Write a mathematical expression for the number of bits in the ROM required in a ROM-and-register implementation of this RRM. (You need not calculate the numerical answer.)

There are 15! possible states, so we would need ceiling(log2(15!)) = 41 state bits. Including the four input bits that indicate which word is being accessed, the ROM would have 2^{45} locations of 41 bits each, for a total of approximately 1442 trillion bits.

F. Is it feasible to build the 15-word RRM above using a ROM and register in today's technology? Explain why or why not.

1442 trillion bits is a bit much even for today's technology. In a .09u technology, a single transistor pulldown in a ROM might require (.09u x .2u) = $.02u^2$, so our ROM would require

about 29 square meters of silicon!