Bill Davis
605.411 Problem Set 7

1. (a) If the cache was direct mapped it would need a 2 bit tag.

   (b)

Line	Tag	Data
0	1 1	Data from line 12
1	0 1	Data from line 5
2	1 1	Data from line 14
3	0 1	Data from line 7

2. This cache contains 32 sets with 4 ways.

   (a)

293824	Maps to set 7, changes the set 7 LRU bits to 110
2948	Maps to set 0, changes the set 0 LRU bits to 110
41728	Maps to set 10, changes the set 10 LRU bits to 110
3072	Maps to set 0, this line is already in the cache, no changes to LRU
1920	Maps to set 0, this line is already in the cache, no changes to LRU
5016	Maps to set 1, changes the set 1 LRU bits to 110
6536	Maps to set 1, this line is already in the cache, no changes to LRU
293764	Maps to set 7, this line is already in the cache, no changes to LRU
4088	Maps to set 0, this line is already in the cache, no changes to LRU
3184	Maps to set 0, this line is already in the cache, no changes to LRU

   (b) In this instance all replacements are correct, while in theory this pseudo LRU replacement scheme could identify a line which is not actually the least recently used.

3. See attached sheet

4. Since C uses row major order, the array will be stored in memory as follows
   0x3EA4000: (0,0) (0,1) (0,2) ... (0,31) (1,0) ... (1,31) ... (31,31)
   0x3EA5000: (32,0) (32,1) ... (32,31) (33,0) ... (33,31) ... (63,31)
   0x3EA6000: (64,0) (64,1) ... (64,31) (65,0) ... (65,31) ... (95,31)

   Where each 4096 byte page can hold 32 consecutive rows from the array. Also since we 32768 bytes available in main memory, we could potentially hold 8 pages or half the entire array in memory at any given time.

   (a) Here the memory access follow the pattern
   (0,0) (1,0) (2,0) ... (0,1) (0,2) ... (348,29) (349,29)

This means there is 1 page fault very 32 accesses. Also, since there are only 8 pages in memory at any given time, by the time we have accessed row 349 and are ready to wrap around back to 0, row 0 has been paged out to disk.

This means there are (11 page faults per column for 30 columns) = 330 page faults

(b) Here the memory accesses follow the pattern
(0,0) (0,1) (0,2) (0,3) ... (1,0) (1,1) ... (349, 28) (349,29)
This means there are only 11 page faults, since we can read every column in a row off the same page.

5. Yes, a program can determine whether it is running on a big-endian machine by storing an integer and checked whether it ended up in the high-order byte or not.
addui $1, $0, 1
sw $1 0($2) //Store word to address
lb $1 0($2) //Load in first byte
andi $t0 $1 1 //If first byte equals 1, then big endian