

**CS 2021: Practice Exam 3 SOLUTION**

Fall 2021

University of Minnesota

Exam period: 20 minutes

Points available: 40

**Problem 1 (10 pts):** Below is an initial memory/cache configuration along with several memory load operations. Indicate whether these load operations result in cache hits or misses and show the state of the cache after these loads complete.

----- SOLUTION -----				
MAIN MEMORY				
Addr	Addr Bits			Value
10	000	10	000	10
14	000	10	100	11
18	000	11	000	12
1C	000	11	100	13
20	001	00	000	20
24	001	00	100	21
28	001	01	000	22
2C	001	01	100	23
30	001	10	000	100
34	001	10	100	101
38	001	11	000	102
3C	001	11	100	103
40	010	00	000	200
44	010	00	100	201
48	010	01	000	202
4C	010	01	100	203
-----+-----+-----+-----+-----				
	Tag	Set	Offset	

DIRECT-MAPPED Cache, 8-byte lines 4 Sets, 8-bit Address = 3-bit tag						
INITIAL CACHE STATE						
Set	V	Tag	Blocks/Line			
			0-3	4-7		
00	1	010	200	201		
01	1	001	22	23		
10	1	000	10	11		
11	0	-	-			

HITS OR MISSES?		
OPEARTION		HIT/MISS?
1. Load 0x48		Miss
2. Load 0x4C		Hit
3. Load 0x24		Miss

FINAL CACHE STATE						
Set	V	Tag	Blocks/Line			
			0-3	8-7	Changed?	
00	1	001	20	21	***	
01	1	010	202	203	***	
10	1	000	10	11		
11	0	-	-			

**Problem 2 (5 pts):** Pyra Midmem read in a free online blog post “Memory for Morons” that there is no need to invest much money in buying RAM. Instead, one can configure the operating system’s virtual memory system to use disk space as main memory leading to a much less expensive computer with a seemingly large memory. Pyra is quite excited about this as some programs she wants to execute fast need a lot of main memory and it would be nice to save some cash. Advise her on any risks or performance drawbacks she may encounter using such an approach.

*SOLUTION: Disks are many orders of magnitude slower than the DRAM that is typically used for main memory. Disks are near the bottom of the memory pyramid offering many gigabytes of storage per dollar at the expense of speed. If Pyra needs speed, she is better off investing in more DRAM as her system will crawl if it attempts to use disk for main memory. In addition, she should consider getting a CPU with a large cache which is more expensive but even faster than DRAM.*

**Problem 3 (15 pts):** Nearby is the definition for `base_scalvec()` which scales a vector by multiplying each element by a number. Write an optimized version of this function in the space provided. Mention in comments why you performed certain transformations.

```
1 int vget(vector_t vec, int idx){
2     return vec.data[idx];
3 }
4 void vset(vector_t vec, int idx, int x){
5     vec.data[idx] = x;
6 }
7 void base_scalevec(vector_t *vec, int *scale){
8     for(int i=0; i < vec->len; i++){
9         int cur = vget(*vec,i);
10        int new = cur * (*scale);
11        vset(*vec,i,new);
12    }
13 }
```

*SOLUTION*

```
1 void opt_scalevec(vector_t *vec, int *scale){
2     // locals to avoid memory access
3     int *data = vec->data, len = vec->len;
4     int scal = (*scale), i;
5     // unroll x2 with duplicate vars to
6     // enable pipelining
7     for(i=0; i < len-2; i+=2){
8         // no function calls - inline bodies
9         // to improve register use
10        int cur0 = data[i+0];
11        int new0 = cur0 * scal;
12        data[i+0] = new0;
13        int cur1 = data[i+1];
14        int new1 = cur1 * scal;
15        data[i+1] = new1;
16    }
17    // cleanup loop
18    for(; i<len; i++){
19        int cur0 = data[i+0];
20        int new0 = cur0 * scal;
21        data[i+0] = new0;
22    }
23 }
```

**Problem 4 (10 pts):** Examine the two functions below which add elements of a row or column vector to all corresponding rows or columns of a matrix. Consider the benchmark timing of these two provided.

1. Explain which of these two functions is faster and **why**.
2. Suggest a way to increase the speed of the slower function with only moderate changes to the code.

*SOLUTION: The addrow() version is clearly faster than addcol() at all sizes and this disparity increases as the sizes of the matrices go up. At the largest size, addrow() takes about 0.2 seconds while addcol() takes 1.5 seconds, a seven-fold difference.*

*The reason is due to the layout of the matrix favoring traversal of rows: each row is contiguous in memory which means loading an element will bring nearby elements in the row into cache. This speeds up their access subsequently. The column version jumps non-contiguously through memory getting much less benefit from cache.*

*Re-writing addcol() to move across rows instead would greatly improve its memory access pattern leading to greater efficiency. This would involve inverting the inner and outer loops to `for(i) / for(j)` as in the row version. This along with slight modifications to the setting would yield speedups*

```
1 ///////////////////////////////////////////////////
2 // SOLUTION
3 // Optimized by inverting loops: no other
4 // changes needed in this case as the logic
5 // for adding on a column is unchanged
6 void matrix_addcolopt(matrix_t mat,
7                         vector_t col) {
8     for(int i=0; i<mat.rows; i++){
9         for(int j=0; j<mat.cols; j++){
10            int elij = MGET(mat,i,j);
11            int veci = VGET(col,i);
12            MSET(mat,i,j, elij + veci);
13        }
14    }
15 }
16 // SIZE  add_row  add_col  add_opt  SPDUP
17 // 512 3.84e-03 7.53e-03 3.71e-03 2.03
18 // 1024 4.21e-03 1.67e-02 4.92e-03 3.39
19 // 2048 1.44e-02 1.09e-01 2.64e-02 4.14
20 // 4096 5.57e-02 5.02e-01 7.43e-02 6.76
21 // 8192 2.28e-01 1.95e+00 2.36e-01 8.25
```