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.

<table>
<thead>
<tr>
<th>Addr</th>
<th>Addr Bits</th>
<th>Value</th>
<th>4 Sets, 8-bit Address = 3-bit tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>000 10 000</td>
<td>10</td>
<td>BLOCKS/LINE</td>
</tr>
<tr>
<td>14</td>
<td>000 10 100</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>000 11 000</td>
<td>12</td>
<td>Set</td>
</tr>
<tr>
<td>1C</td>
<td>000 11 100</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>001 00 000</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>001 00 100</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>001 01 000</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>2C</td>
<td>001 01 100</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>001 10 000</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>001 10 100</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>001 11 000</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>3C</td>
<td>001 11 100</td>
<td>103</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>010 00 000</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>010 00 100</td>
<td>201</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>010 01 000</td>
<td>202</td>
<td></td>
</tr>
<tr>
<td>4C</td>
<td>010 01 100</td>
<td>203</td>
<td></td>
</tr>
</tbody>
</table>

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.
**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.

```c
int vget(vector_t vec, int idx){
    return vec.data[idx];
}

void vset(vector_t vec, int idx, int x){
    vec.data[idx] = x;
}

void base_scalevec(vector_t *vec, int *scale){
    for(int i=0; i < vec->len; i++){
        int cur = vget(*vec,i);
        int new = cur * (*scale);
        vset(*vec,i,new);
    }
}
```

**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.

```c
// add given row to each row of mat
void matrix_addrow_vec(matrix_t mat, vector_t row) {
    for(int i=0; i<mat.rows; i++){
        for(int j=0; j<mat.cols; j++){
            int elij = MGET(mat,i,j);
            int vecj = VGET(row,j);
            MSET(mat,i,j, elij + vecj);
        }
    }
}

// add given col to each column of mat
void matrix_addcol_vec(matrix_t mat, vector_t col) {
    for(int j=0; j<mat.cols; j++){
        for(int i=0; i<mat.rows; i++){
            int elij = MGET(mat,i,j);
            int veci = VGET(col,i);
            MSET(mat,i,j, elij + veci);
        }
    }
}
```

```
// BENCHMARK TIMING:
// SIZE addrow addcol
// 512 2.9040e-03 5.5230e-03
// 1024 5.9290e-03 1.3160e-02
// 2048 1.3809e-02 9.9269e-02
// 4096 5.0853e-02 3.6760e-01
// 8192 2.0867e-01 1.4719e+00
```