**Question 6: Cache in While You Can** (17 points, 26 Minutes)

Consider a single 4KiB cache with 512B blocks and a write-back policy. Assume a 32-bit address space.

a) If the cache were direct-mapped,

\[ \text{sets} \]

\[ \# \text{ of rows? } \] ________ \[ \# \text{ of offset bits? } \] ________

b) If the cache were 4-way set associative,

\[ \# \text{ of tag bits? } \] ________ \[ \# \text{ of index bits? } \] ________ \[ \# \text{ of bits per cache slot? } \] ________

Consider an array of the following `location` structs:

```c
typedef struct {
    ... // some undefined number of other struct members
    int visited;
    int danger;
} location;

location locs[NUM_LOCS];
```

Here's a piece of code that counts the number of places we've visited. Assume this gets executed somewhere in the middle of our program, that `count` is held in a register, and the size of the array is greater than 4 KiB.

```c
for(int i = 0; i < NUM_LOCS; i++)
    if(locs[i].visited) count++;
```

c) What's the fewest possible number of bytes written to main memory? ________
d) What's the greatest possible number of bytes written to main memory? ________

Now consider if we store the `visited` and `danger` information in individual arrays instead:

```c
int visited[NUM_LOCS];
int danger[NUM_LOCS];
```

e) This way, the cache can exploit better ________________ for the above task.

We can expect a ________________ (higher or lower) miss rate
because of the change in the number of ________________ (type of cache miss) misses.
Consider the following code with $\text{NUM LOCS} > 2^{10}$.

```java
for(int i = 0; i < \text{NUM LOCS}; i++)
    if(visited[i] && danger[i] > 5) count++;
```

Two memory accesses are made per iteration: one into \text{visited}, the other into \text{danger}. Assume that the cache has no valid blocks initially. You are told that in the worst case, the cache has a miss rate of 100%. Consider each of the following possible changes to the cache individually.

f) Mark each as E, if it eliminates the chances of this worst-case scenario miss rate, R if it reduces the chances, or N if it’s not helpful.

- More sets, same block size, same associativity
- Double associativity, half block size, same total cache size
- Everything stays the same but use a write-through policy instead
Login: cs61c–____

3) “Cache, money. Dollar bills, y’all.” (24 min, 15 pts)

Suppose we have a standard 32-bit byte-addressed MIPS machine, a single direct-mapped 32KiB cache, a write-through policy, and a 16B block size.

a) Give the T:l:O breakup. ______________________
b) How many bits are there per row on the cache? __________________________________________

Use the C code below and the description of the cache above to answer the questions that follow it. Suppose that the only memory accesses are accesses and stores to arrays and that all memory accesses in the code are valid. Assume A starts on a block boundary (byte 0 of A in byte 0 of block).

```c
#define NUM_INTS 32
#define OFFSET 8192  // 8192 = 2^13

int rand(int x, int y); // returns a random integer in the range [x, y)

int main(){
    int A[NUM_INTS + OFFSET]; // Assume A starts on a block boundary

    // START LOOP 1
    for (int count = 0; count < NUM_INTS; count += 1) { // count by 1s
        A[count] = count; // ACCESS #1
        A[count + OFFSET] = count+count; // ACCESS #2
    }
    // END LOOP 1

    // START LOOP 2
    for (int count = 0; count < NUM_INTS; count += 4) { // count by 4s now
        for (int r = 0; r < 4; r++) {
            printf("%d", A[rand(count, count+4)]);
        }
    }
    // END LOOP 2
}
```

c) Hit rate for Loop 1? _______________ What types of misses are there? ____________________________
d) Hit rate for Loop 2? _______________ What types of misses are there? ____________________________

Questions (e), (f), and (g) below are three independent variations on the original code & settings.

e) If the cache were 2-way set associative, what would be the hit rate for Loop 2? ________________
   (assume the standard LRU replacement policy)

f) If instead we removed the line labeled ACCESS #2, what would be the hit rate for Loop 2? _______

g) Instead, what’s the smallest we could shrink OFFSET to maximize our Loop 2 hit rate? ___________
   (assume we still need to maintain the same functionality)
M2) Cache Money, y'all (10 pts, 20 min)

This C code runs on a 32-bit MIPS machine with 4 GiB of memory and a single L1 cache. Vectors A, B live in different places of memory, are of equal size (n is a power of 2 and a [natural number] multiple of the cache size), block aligned. The size of the cache is C, a power of 2 (and always bigger than the block size, obviously).

```
// sizeof(uint8_t) = 1
SwapLeft(uint8_t *A, uint8_t *B, int n) {
    uint8_t tmp;
    for (int i = 0; i < n; i++) {
        tmp = A[i];
        A[i] = B[i];
        B[i] = tmp;
    }
}

// sizeof(uint8_t) = 1
SwapRight(uint8_t *A, uint8_t *B, int n) {
    uint8_t tmpA, tmpB;
    for (int i = 0; i < n; i++) {
        tmpA = A[i];
        tmpB = B[i];
        B[i] = tmpA;
        A[i] = tmpB;
    }
}
```

Let's first just consider the SwapLeft code for parts (a) and (b).

a) If the cache is direct mapped and the best hit:miss ratio is “H:1", what is the block size in bytes? _____

b) What is the worst hit:miss ratio? _____:_____ 

c) Fill in the code for SwapRight so that it does the same thing as SwapLeft but improves the (b) hit:miss ratio. You may not need all the blanks.

d) If the block size (in bytes) is a, what is the worst hit:miss ratio for SwapRight? ______:______

e) We next change the cache to be 2-way set-associative, and let's go back to just considering SwapLeft. What is the worst hit:miss ratio for SwapLeft with the following replacement policies? The cache size is C (bytes), the block size is a (bytes), LRU = Least Recently Used, MRU = Most Recently Used.

<table>
<thead>
<tr>
<th>LRU and an empty cache</th>
<th>MRU and a full cache</th>
</tr>
</thead>
<tbody>
<tr>
<td><em><strong><strong>:</strong></strong></em></td>
<td><em><strong><strong>:</strong></strong></em></td>
</tr>
</tbody>
</table>
d) Ash Ketchum has six slots in his party, each of which can hold a single Pokémon. Additionally, Ash has access to a PC (personal computer) which holds the rest of the Pokémon he owns. Essentially, his party acts as a “cache” for accesses to the PC (the “memory”).

i. Each slot in Ash’s party can hold any Pokémon. What kind of cache is this analogous to? (Circle one)

- Set-associative
- Write-back
- Fully Associative
- Direct Mapped
- Write-through

ii. Ash’s party exploits ____________ locality but not ____________ locality.

Explain in one sentence:

e) Fill in the **recursive** definition of `strcmp()` below. `strcmp` returns 0 if the contents of the strings are identical, otherwise it returns the difference in the ASCII representations of the first non-matching characters (negative number if \( s_1 < s_2 \) lexicographically).

```c
int strcmp(char *s1, char *s2) {
    if ( ____________________________________________ )
        return ______________________________________;
    return (*s1 - *s2);
}
```

f) MIPS already has a 64-bit architecture, so it’s just a matter of time before MIPS128 is released. Let’s help them out with their 128-bit instruction format design. Answer the following questions **independently** based on the MIPS32 design taught in class.

i. If we doubled the opcode field size and quadruple the register field sizes, how many instructions **forward** can we reach with a single branch? Answer in IEC.

ii. If we quadruple the total number of I and J format instructions, what fraction of memory could we reach with a single jump instruction? Feel free to leave as a power of 2.

iii. If we want to keep the same number of R, I, and J instructions, what is the maximum number of registers our architecture can support? Answer in IEC.
Question 4: Caches (11 pts)

We have a 64 KiB address space and two different caches. Both are 1 KiB, direct-mapped caches with random replacement and write-back policies. Cache X uses 64 B blocks and Cache Y uses 256 B blocks.

a) Calculate the TIO address breakdown for Cache X:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
</table>

b) During some part of a running program, Cache Y’s management bits are as shown below. Four options for the next two memory accesses are given (R = read, W = write). Circle the option that results in data from the cache being written to memory.

<table>
<thead>
<tr>
<th>Line</th>
<th>Slot</th>
<th>Valid</th>
<th>Dirty</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1000</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1010</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1110</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0000</td>
</tr>
</tbody>
</table>

(1) R 0x4C00, W 0x5C00
(2) W 0x5500, W 0x7A00
(3) W 0x2300, R 0x0F00
(4) R 0x3000, R 0x3000

c) The code snippet below loops through a character array. Give the value of LEAP that results in a Hit Rate of 15/16 for Cache Y.

```c
#define ARRAY_SIZE 8192
char string[ARRAY_SIZE]; // &string = 0x8000
for(i = 0; i < ARRAY_SIZE; i += LEAP) {
    string[i] |= 0x20; // to lower
}
```

d) For the loop shown in part (c), let LEAP = 64. Circle ONE of the following changes that increases the hit rate of Cache X:

- Increase Block Size
- Increase Cache Size
- Add a L2$
- Increase LEAP

e) For the following cache access parameters, calculate the AMAT. All miss and hit rates are local to that cache level. Please simplify and include units.

<table>
<thead>
<tr>
<th>L1$ Hit Time</th>
<th>L1$ Miss Rate</th>
<th>L2$ Hit Time</th>
<th>L2$ Hit Rate</th>
<th>MEM Hit Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 ns</td>
<td>40%</td>
<td>20 ns</td>
<td>95%</td>
<td>400 ns</td>
</tr>
</tbody>
</table>
Question 8: Caches (10 pts)

We are using a 20-bit byte addressed machine. We have two options for caches: Cache A is fully associative and Cache B is 4-way set associative. Both caches have a capacity of 16 KiB and 16 B blocks.

a) Calculate the TIO address breakdown for Cache A:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>

b) Below is the initial state of one set (four slots) in Cache B. Each slot holds 2 LRU bits, with 0b00 being the most recently used and 0b11 being the least recently used. Circle ONE option below for two memory accesses that result in the final LRU bits shown and only one block replacement.

<table>
<thead>
<tr>
<th>Index</th>
<th>Initial</th>
<th>Final</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001 1110</td>
<td>0 0110 1010 00 → 10</td>
<td>0 1010 1100 11</td>
</tr>
<tr>
<td></td>
<td>1 0000 0001 10</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td>2 0101 0101 01</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>3 1010 1100 11</td>
<td>01</td>
</tr>
</tbody>
</table>

(1) 0x019D0, 0xAD9D0  (2) 0xAC9E0, 0x129E0  (3) 0xAD9D0, 0x019D0  (4) 0x129E0, 0xAC9E0

c) For the code given below, calculate the hit rate for Cache B assuming that it starts cold.

```c
#define ARRAY_SIZE 8192
int int_arr[ARRAY_SIZE];    // &int_arr = 0x80000
for (int i = 0; i < ARRAY_SIZE / 2; i++) {
    int_arr[i] *= int_arr[i + ARRAY_SIZE / 2];
}
```

d) For each of the proposed changes below, write U for “increase”, N for “no change”, or D for “decrease” to indicate the effect on the hit rate of Cache B for the loop shown in part (c):

- Direct-mapped ______  Increase cache size ______
- Double ARRAY_SIZE _____  Random block replacement _____

e) Calculate the AMAT for a multi-level cache given the following values. Don’t forget units!

<table>
<thead>
<tr>
<th>L1$ HT</th>
<th>L1$ MR</th>
<th>L2$ HT</th>
<th>GMR</th>
<th>MEM HT</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 ns</td>
<td>20%</td>
<td>25 ns</td>
<td>5%</td>
<td>500 ns</td>
</tr>
</tbody>
</table>