Scenario 2:

1): Venus setup for Scenario 2 showing Program parameters, Cache parameters etc

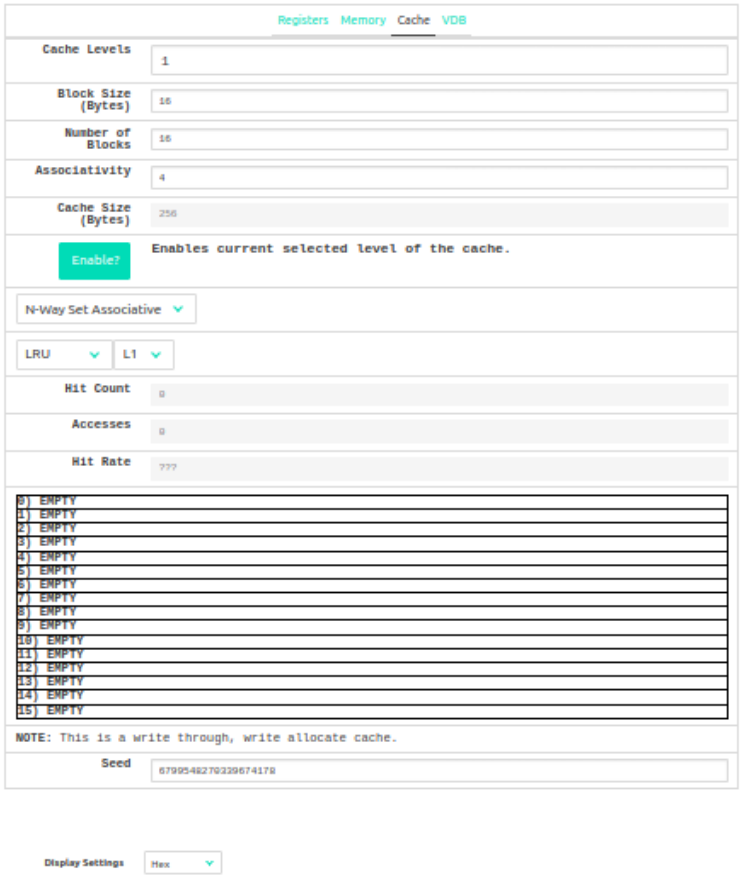
Active File: null Save Close

```
#      for (index = 0; index < arraysize; index += stepsize) {
#        if(option==0)
#          array[index] = 0;                    // Option 0: One cache access - write
#        else
#          array[index] = array[index] + 1; // Option 1: Two cache accesses - read AND write
#      }
#    }

.data
array:  .word   2048                            # max array size specified in BYTES (DO NOT CHANGE)

.text
##############################################################################################
main:   li  a0, 256     # array size in BYTES (power of 2 < array size)
    li  a1, 2       # step size  (power of 2 > 0)
    li  a2, 1       # rep count  (int > 0)
    li  a3, 1       # 0 - option 0, 1 - option 1
# You MAY change the code above this section
##############################################################################################
```

Cache parameters are:



……

2): Answers to tasks 1, 2, 3, 4

1. How many **memory accesses** are there per iteration of the **inner loop** (not the one involving Rep Count)?

   There are 2 **memory accesses** per iteration of the inner loop because in a single iteration firstly we are loading by lw t0, 0(s0) which is one access then we are storing by sw t0, 0(s0) which is another access. As it is write through cache so it will update the cache and the memory at the same time. So in “write” case there will also be a memory access.

2. What is the **repeating hit/miss pattern**? Write your answer in the form "mmhhmh" and so on, where your response is the **shortest** pattern that gets repeated.

   mhhh is the shortest repeating pattern

3. Keeping everything else the same, what does our hit rate approach as Rep Count goes to infinity? Try it out by changing the appropriate program parameter and letting the code run! Write your answer as a decimal.

approaches to 1.0 because cache is fully filled with all the elements of the array. So accessing them again will always result a hit.

4. Suppose we have a program that iterates through a very large array (i.e. way bigger than the size of the cache) Rep Count times. During each Rep, we map a different function to the elements of our array (e.g. if Rep Count = 1024, we map 1024 different functions onto each of the array elements, one per Rep). For reference, in this scenario, we just had one function (incrementation) and one Rep.

Instead, we should try to access **portion** of the array at a time and apply all of the **function** to that **portion** so we can be completely done with it before moving on, thereby keeping that **portion** hot in the cache and not having to circle back to it later on! (The 1st, 3rd, and 4th blanks should be the same. It's not some vocabulary term you should use to fill them in. It's more of an idea that you should have.)

……