SOLUTIONS:

BLUE FRAGMENT:

- Write-back, write allocate:
    - D$ Accesses: 48
    - D$ Misses: 4

  Discussion: Note that we have 48 accesses, 32 corresponding to load operations and 16 to store operations. As for the misses, all the accesses to the C array hit in the D$, as they were brought into the D$ in the previous loop (write-allocate). Instead, the accesses to the R array generate 1 miss every 4 accesses, as the elements of this array have not been brought into the D$ yet, and, given that the D$ uses a *write allocate* policy, they are being brought now.

- Write-through, no write allocate:
    - D$ Accesses: 48
    - D$ Misses: 24

  Discussion: In this case, we have the same number of accesses as before. Instead we have now 24 misses; 8 of them are due to the read accesses to the C array, which, given that the D$ uses a *no write allocate* policy, generate 1 miss every 4 accesses. The remaining 16 misses are due to the write accesses to the R array, which again, given that the D$ uses a *no write allocate* policy, generate 1 miss per accesses.

GREEN FRAGMENT:

- Write-back, write allocate:
    - D$ Accesses: 512
    - WBs: 48

Array D will occupy the whole D$ (the D$ has 2028B and the D array has 512 integers). Thus, we have to write back all dirty blocks, which are the blocks from arrays C and R. Note that arrays A and B where modified but then the D$ was emptied, writing back those arrays.

- Write-through, write allocate:
    - D$ Accesses: 512
    - WB: 0

In this case, all stores updated memory when they were executed, thus we do not need to write anything back.