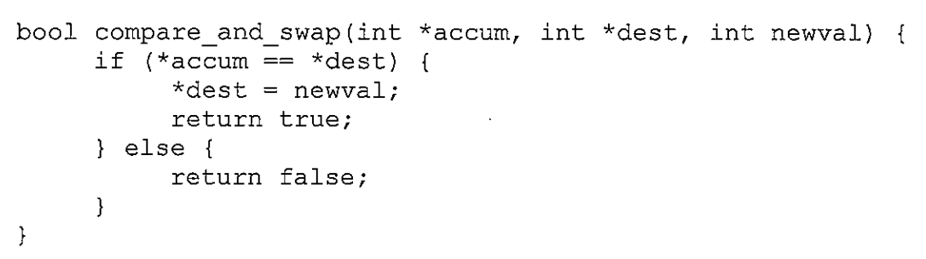
**CS3014 – Concurrent Systems**

**2018 Exam Solutions**

**Question 1**

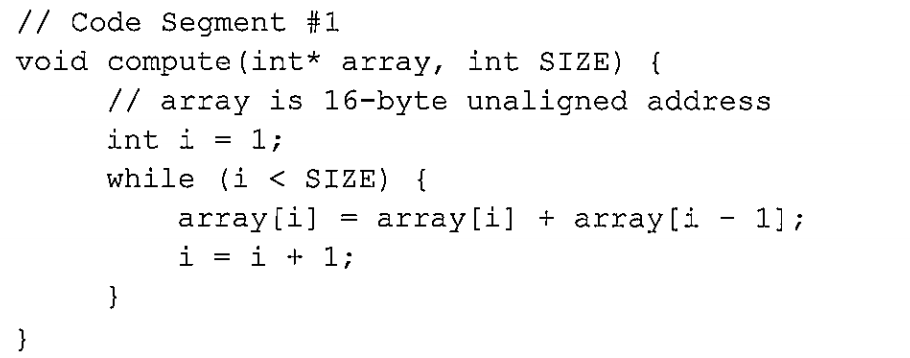
*a) Compare-and-Swap (CAS) is an atomic instruction used in multithreading to achieve synchronization. It compares the contents of a memory location with a given value, and only if they are the same, modifies the contents of that memory location to a new given value. Explain how this can be used to create a lock between threads.*

**

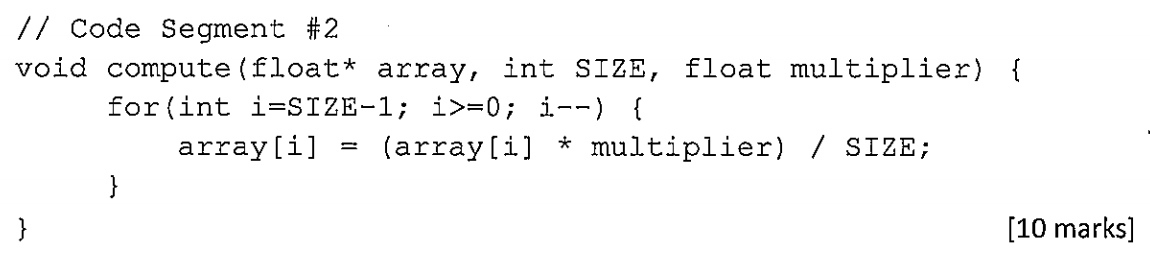
Compare and swap is done in a single atomic operation. The atomicity guarantees that the new value is calculated based on up to date information; if the value has been updated by another thread in the meantime, the write would fail. The result of the operation indicates whether the write was performed or not.

Algorithms based around CAS typically read some key from a memory location and remember the old value. Using this old value a new value is then computed. Then, using CAS, where the comparison checks the location still being equal to the old value. If this is true, no other thread has updated the value and as a result it is okay to perform the write.

*b) Examine and state whether each of the following blocks of code can be parallelised effectively using SSE (x86). If not, clearly state why. If the code can be parallelised, use SSE intrinsics to vectorise it. Write a short note about the parallelisation strategy you used.*

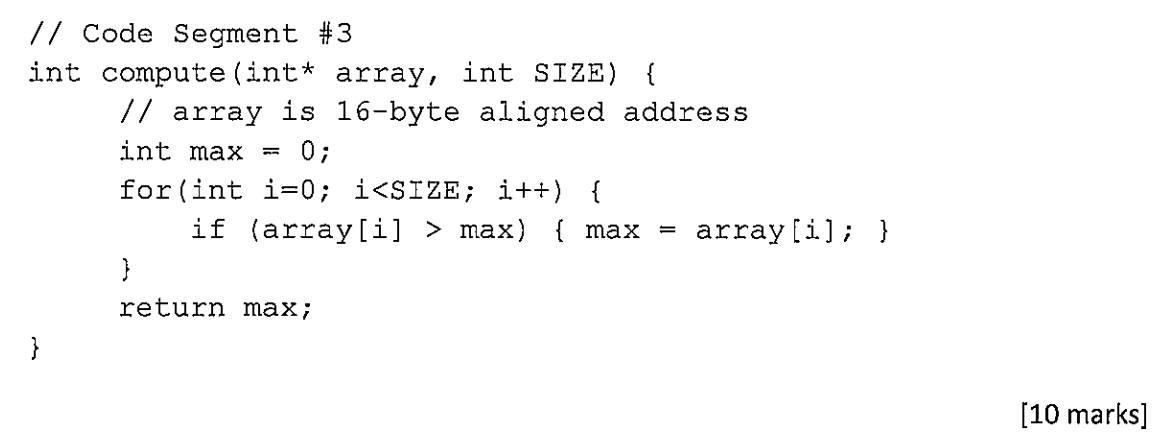
**

The above code cannot be vectorised as the array addition step depends on the result of the previous addition. This dependency stems all the way from a[0] up to a[SIZE-1]. For each I we are replacing array[i] with a sum of an old value which would also have been calculated.



The above code can be vectorised as there is no underlying dependency within each operation within the for loop. We can reduce the number of loops by a factor of 4 by extracting 4 x 32-bit float values at once into an SSE vector. We can then perform the multiplication and division on 4 elements of the array simultaneously and store the result from the four operations back into the array. The for loop must also be decremented by 4 on each loop to account for this.





The above code can be vectorised as there is no underlying dependency within each operation within the for loop. We can perform 4 max operations at a time thus reducing the for loop by a factor of 4. Once we have finished, we are left with a vector containing 4 possible maximums. We then extract the max element from this vector to get a final result.

