Exercise 1.

# **Analysis of Data Dependency**

S1: a = f1();

S2: b = f2(a); True dependence between S1 and S2

S3: c = f3(a); True Dependence between S1 and S3

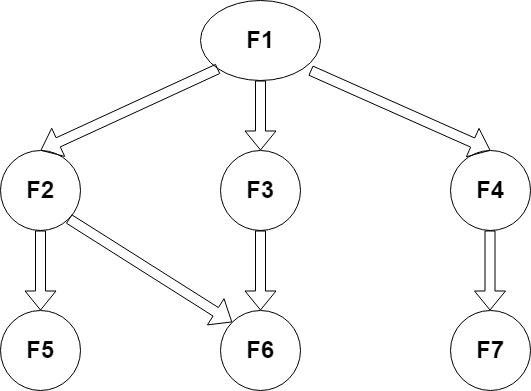
S4: a = f4(a); Anti-Dependence between S2,S3 and S4, also there is Output dependence between S1 and S4

# **Parallelization Of the Program**

To give an extra room for the possibility parallelization in addition to the existing ones which are F2 and F3, I removed an anti-dependence occurring for the F4 which I added a new variable for which f4 is now stored, that is changing a=f4(a) to something like d=f(4), but in each case, F1 function must be executed first.

With the usage of condition variables, 3 threads were used Thread 1 for F1, F3, F6 Thread 2 for F2 and F5, Thread 3 was used for F4 and F7.

Additional flag was used for F6 function and the reason for that is to due to its dependence on F2.



The execution time of the functions if each function is 1s will be 3 s. The answer is because 3 threads were used, so I used the time of thread 1 which is the highest of them all which is 3s.

## SOME OBSERVATIONS AND CHALLENGES EXPERIENCED

After successfully parallelizing the program, I happened to get a different result which is 16588111872. With much time spent on analysis of the program trying to figure out the reason why my answer differs from the sequential program which was given a value of 16588111878.6845608. I then discovered that it was from the arithmetic of the final answer, instead of a sequence of F5 + F7 +F6 in my code it was F5+ F6 + F7.

In my opinion the reason for this is due to the addition of floating point reaching a dedicated value for which the computer will automatically approximate a number. Such case I think occurred from the addition of F5 and F6 but since F7 is actually a negative number, hence, F5+F7 would result to a number less than the dedicated value for approximation qualification.