Algorithm:

Calculation process can be divided into 3 parts: population stage, freeing stage and execution stage.

Population stage:  
 1) finding which resources the instruction uses  
 2) allocation resources by pushing an instruction id into the resource’s queue  
 3) incrementation of instruction\_count

Freeing stage:  
 1) instruction that accessed the resource are popped from the queue

Execution stage:  
 1) checking if instruction resource requests were satisfied, if they were, execute the instruction  
 2) if requirements have not been met, skip the instruction and check the next one

About “ring buffer”  
 I don’t know if the people checking the programme have boost libraries installed, so instead I made a simple container, that, with some work-arounds, will function similar to a ring buffer in this case. The same goes for the use of unordered map, instead of boost’s bidirectional map.

Explaining the clock\_cycle gap  
 Here’s an example:  
After the first cycle, which will issue instructions 1 and 4, and the second cycle that will issue instruction 2 and 5, in the third cycle however instruction 3 tries to read from the register 3, and instruction 6 tries to write to register 3, creating a possible hazard, thus, in my implementation, execution of instruction 6 would be delayed, shifting the amount of clock cycles.