In this post, our tour through the c++ memory model goes one step deeper. Until now, the posts were only about the atomicity of the atomic data types but now we deal with the synchronisation and ordering constraints of the operations.

You can not configure the atomicity of an atomic data type, but you can adjust very accurately the synchronisation and ordering constraints of atomic operations. A leverage, which is unique to C++. That's not possible in the C#''s or Java's memory model.

he six variants of the C++ memory model

C++ has six variants of the memory model. The default for atomic operations is std::memory\_order\_seq\_cst. But you can explicitly specify one of the other five. But what has C++11 to offer?

enum memory\_order{

memory\_order\_relaxed,

memory\_order\_consume,

memory\_order\_acquire,

memory\_order\_release,

memory\_order\_acq\_rel,

memory\_order\_seq\_cst

}

It helps a lot to answer two questions, to get a system into the six memory models.

1. For which type of atomic operations should you use the memory model?
2. Which synchronisation and ordering constraints are defined by the memory model?

The rest of this post is about answering these questions. So what are the types of atomic operations?

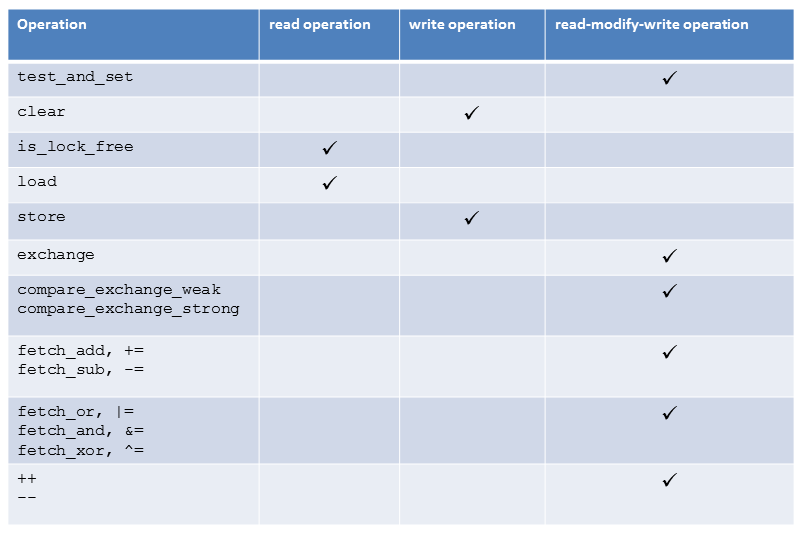
Types of atomic operations

The memory model deals with reading and/or writing atomic operations.

* **read operation:**memory\_order\_acquire and memory\_order\_consume
* **write operation:**memory\_order\_release
* **read-modify-write operation:**memory\_order\_acq\_rel and memory\_order\_seq\_cst

memory\_order\_relaxed defines no synchronisation and ordering constraints. So it will not fit in this taxonomy.

The table orders the atomic operations based on their reading and/or writing characteristics.



In case you use an atomic operation atomVar.load(5) with a memory model, that is designed for a write or read-modify-write operation, the write part has no effect. So an atomVar.load(5,std, std::memory\_order\_acq\_rel) is equivalent to an atomVar.load(5,std::memory\_order\_acquire), an atomVar.load(5, std::memory\_order\_release) is equivalent to an atomVar.load(5, std::memory\_order\_relaxed).

The different synchronisation and ordering constraints

There are three different types synchronization and ordering constraints in C++11:

* **Sequential consistency**: memory\_order\_seq\_cst
* **Acquire-release**: memory\_order\_consume, memory\_order\_acquire, memory\_order\_release and memory\_order\_acq\_rel
* **Relaxed**: memory\_order\_relaxed

While the sequential consistency establishes a global order between threads, the acquire-release semantic establishes an ordering between read and write operations on the same atomic variable on different threads. The relaxed semantic only guarantees that operations on the same atomic data type in the same thread can not be reordered. That guarantee is called [modification order consistency.](http://en.cppreference.com/w/cpp/atomic/memory_order) But other threads can see this operation in a different order.