Concurrency: Multi-core Programming & Data Processing

Lab 5

-- Memory Barriers --

Do not confuse...

- Most important concepts:
  - **Written code =/= machine code**
  - **Memory barriers =/= Java built-in barriers**
- What you write is <u>not</u> what you get (most of the times)…
- Why?

What goes on under the hood?

What goes on under the hood?

- Compiler can reorder the statements

What goes on under the hood?

- Compiler can reorder the statements
- Processor can reorder the statements (corresponding machine instructions, that is), or execute them at the same time

What goes on under the hood?

- Compiler can reorder the statements
- Processor can reorder the statements (corresponding machine instructions, that is), or execute them at the same time
- Memory can reorder the statements (more precisely, the order of writes and reads)

What goes on under the hood?

- Compiler can reorder the statements
- Processor can reorder the statements (corresponding machine instructions, that is), or execute them at the same time
- Memory can reorder the statements (more precisely, the order of writes and reads)
- Any of them may interleave the machine-level effect of two consecutive actions
- Any of them may cause the memory cells not to be updated immediately (or at all...)

What goes on under the hood?

- Compiler can reorder the statements
- Processor can reorder the statements (corresponding machine instructions, that is), or execute them at the same time
- Memory can reorder the statements (more precisely, the order of writes and reads)
- Any of them may interleave the machine-level effect of two consecutive actions
- Any of them may cause the memory cells not to be updated immediately (or at all...)

- ...and all this is fine as long as the execution obeys "as-if-serial" semantics and we are single-threaded

But then concurrency happens...

- We need ordering limitations on memory operations => **memory barriers** (aka **fences**)
- Direct control of CPU with its cache, write-buffer, read-buffer
- Serialize pending memory operations

But then concurrency happens...

- We need ordering limitations on memory operations => **memory barriers** (aka **fences**)
- Direct control of CPU with its cache, write-buffer, read-buffer
- Serialize pending memory operations
- Enforce **visibility** (sounds familiar?)

Focus on Java

- Interpreted language: generates bytecode, interprets bytecode
- Also has **Just-In-Time (JIT)** compiler: bytecode of hot methods compiled to machine code, machine code run directly for them
  - Optimizes quite a lot the execution
- To see assembler code (needs hsdis disassembler plugin):

```
  java -XX:+UnlockDiagnosticVMOptions -XX:+PrintAssembly
Application
```

- To check if a method was compiled: `-XX:+PrintCompilation`
- Interactive graphical interface: JITWatch

Exercise

- Install hsdis plugin or JITWatch on your system
- Download MemBarrier.java from lab5/exercises on ILIAS
- Modify it to use Atomic variables instead of volatile (call it MemBarrierAtomic.java)
- Run and disassemble (generate log file)
- Write a brief README with the observations (compare the use of volatile with Atomic), command line you used, OS, etc.
- Submit ZIP archive with new java file, log and README