



Модель памяти .NET

Валерий Петров

Обо мне

- Окончил МатМех СПбГУ
- Сейчас работаю в компании Sidenis
- Увлекаюсь concurrency и всякими «кишочками»
- Нравится узнавать новое
- Нравнодушен к качественному коду

DotNext 2016 Piter



DotNext SPb 2016

@goldshn

<https://s.sashag.net/dnspp12>

The C++ and CLR Memory Models

Sasha Goldshtein
CTO, Sela Group
@goldshn

Вопрос:

Q: Компьютер выполняет программу, которую Вы написали?

Вопрос:

Q: Компьютер выполняет программу, которую Вы написали?

A: Нет.

Компилятор, JIT и CPU умеют **оптимизировать!**

Почему они это делают?

Почему они это делают?

Раньше инструкции исполнялись последовательно

Cycle	1	2	3	4	5	6	7	8	9
Instr ₁	Fetch	Decode	Execute	Write					
Instr ₂					Fetch	Decode	Execute	Write	
Instr ₃									Fetch

<http://www.slideshare.net/nithilgeorge/2010-1002-intro-to-microprocessors1>

Почему они это делают?

Затем появился конвейер

Cycle	1	2	3	4	5	6	7	8	9
Instr ₁	Fetch	Decode	Execute	Write					
Instr ₂		Fetch	Decode	Execute	Write				
Instr ₃			Fetch	Decode	Execute	Write			
Instr ₄				Fetch	Decode	Execute	Write		
Instr ₅					Fetch	Decode	Execute	Write	
Instr ₆						Fetch	Decode	Execute	Write

<http://www.slideshare.net/nithilgeorge/2010-1002-intro-to-microprocessors1>

Почему они это делают?

Но инструкции всё равно исполнялись в порядке следования

Cycle	1	2	3	4	5	6	7	8	9
Instr ₁	Fetch	Decode	Execute			Write			
Instr ₂		Fetch	Decode	Wait		Execute	Write		
Instr ₃			Fetch	Decode	Wait		Execute	Write	
Instr ₄				Fetch	Decode	Wait		Execute	Write
Instr ₅					Fetch	Decode	Wait		Execute
Instr ₆						Fetch	Decode	Wait	

<http://www.slideshare.net/nithilgeorge/2010-1002-intro-to-microprocessors1>

Почему они это делают?

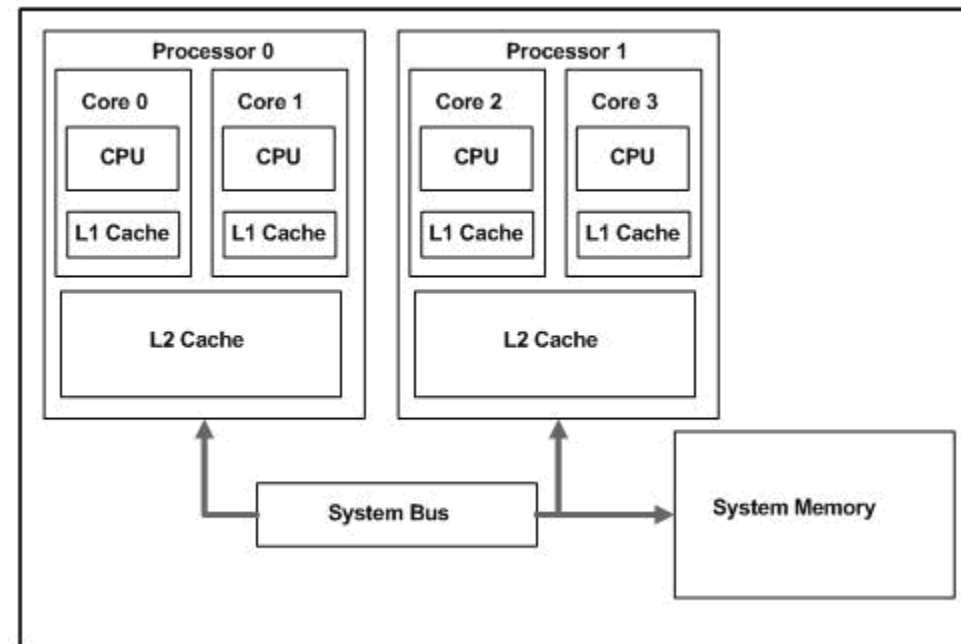
Затем произошло массовое внедрение out-of-order execution

Cycle	1	2	3	4	5	6	7	8	9
Instr ₁	Fetch	Decode	Execute			Write			
Instr ₂		Fetch	Decode	Wait		Execute	Write		
Instr ₃			Fetch	Decode	Execute	Write			
Instr ₄				Fetch	Decode	Wait	Execute	Write	
Instr ₅					Fetch	Decode	Execute	Write	
Instr ₆						Fetch	Decode	Execute	Write

<http://www.slideshare.net/nithilgeorge/2010-1002-intro-to-microprocessors1>

Почему они это делают?

- Доступ к памяти (долго)
- Инвалидация кэша (дорого)
- Производительность (IPC)



https://software.intel.com/sites/default/files/m/d/4/1/d/8/286501_286501.gif

Что могут сделать компилятор/JIT/CPU?

- **Loop Read Hoisting** - while (true)
- **Read Elimination** – кэширование в регистрах (иногда даже volatile не спасает – в .NET Framework тоже бывали баги)
- **Read Introduction** – устранение локальной переменной (может выстрелить в мире concurrency)
- И ещё много чего, что позволяет спецификация

Loop Read Hoisting

```
private bool flag = true;

[Test]
public void LoopReadHoistingTest()
{
    Task.Run(() => { flag = false; });

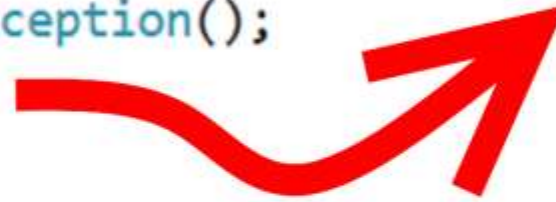
    while (flag)
    {
    }
}
```

Read Elimination

```
public int ReadElimination(int x, int y)
{
    if (x == 0)
        throw new ArgumentException();

    var result = y + 42;
    result += x;

    return result;
}
```



```
public int ReadElimination(int x, int y)
{
    var tmp = x;
    if (tmp == 0)
        throw new ArgumentException();

    var result = y + 42;
    result += tmp;

    return result;
}
```

Read Introduction

```
private object someObject;
```

```
private void PrintObj()
```

```
{
    var localReference = someObject;
    if (localReference != null)
        Console.WriteLine(localReference.ToString());
}
```

```
private void Uninitialize()
```

```
{
    someObject = null;
}
```

```
private object someObject;
```

```
private void PrintObj()
```

```
{
    if (someObject != null)
        Console.WriteLine(someObject.ToString());
}
```


```
private void Uninitialize()
```

```
{
    someObject = null;
}
```


А как вам такая перестановка?

```
public void Initialize()
{
    Instance = new SomeClass();
    initialized = true;
}
```

```
public void Initialize()
{
    initialized = true;
    Instance = new SomeClass();
}
```



Модель памяти

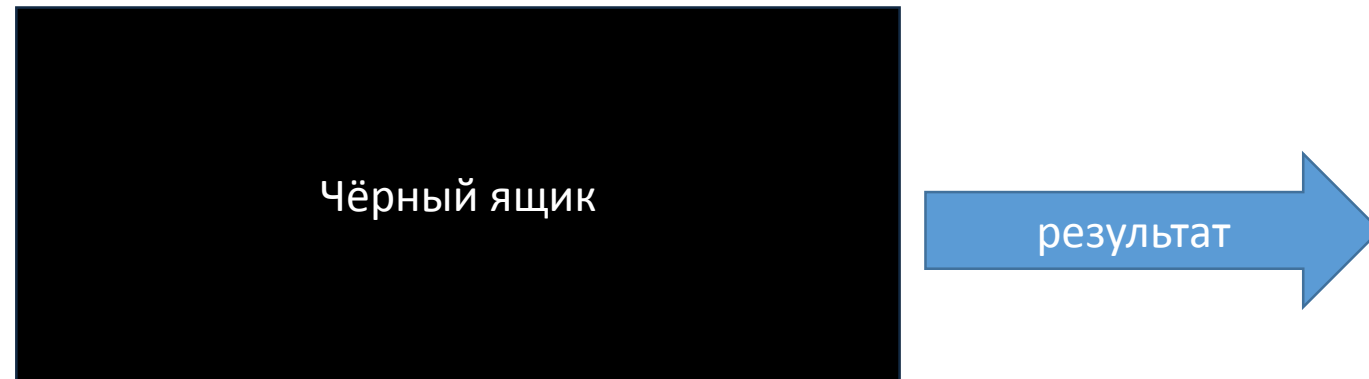
- In computing, a **memory model** describes the interactions of threads through memory and their shared use of the data.
(Wikipedia)
- ECMA-335 и ECMA-334
- Слабые и сильные модели памяти
- Спецификации на модель памяти Microsoft CLR не существует!

Разные архитектуры

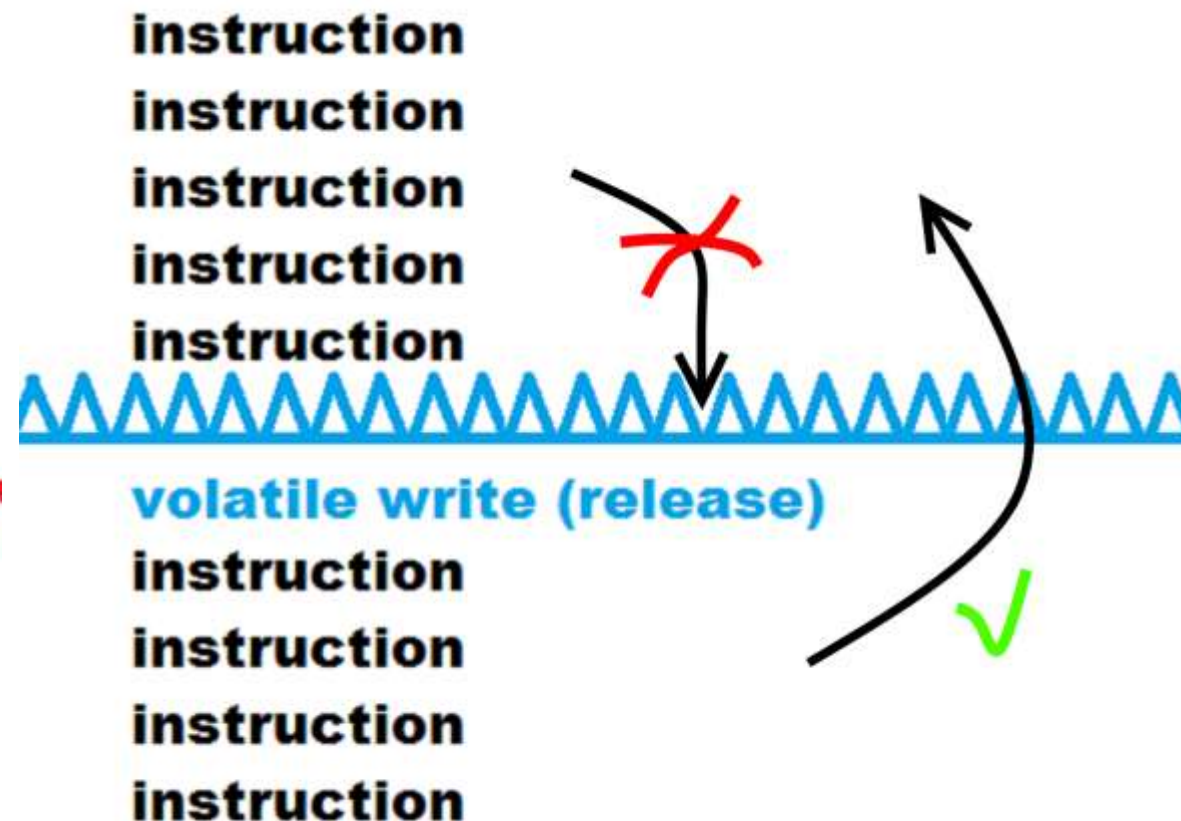
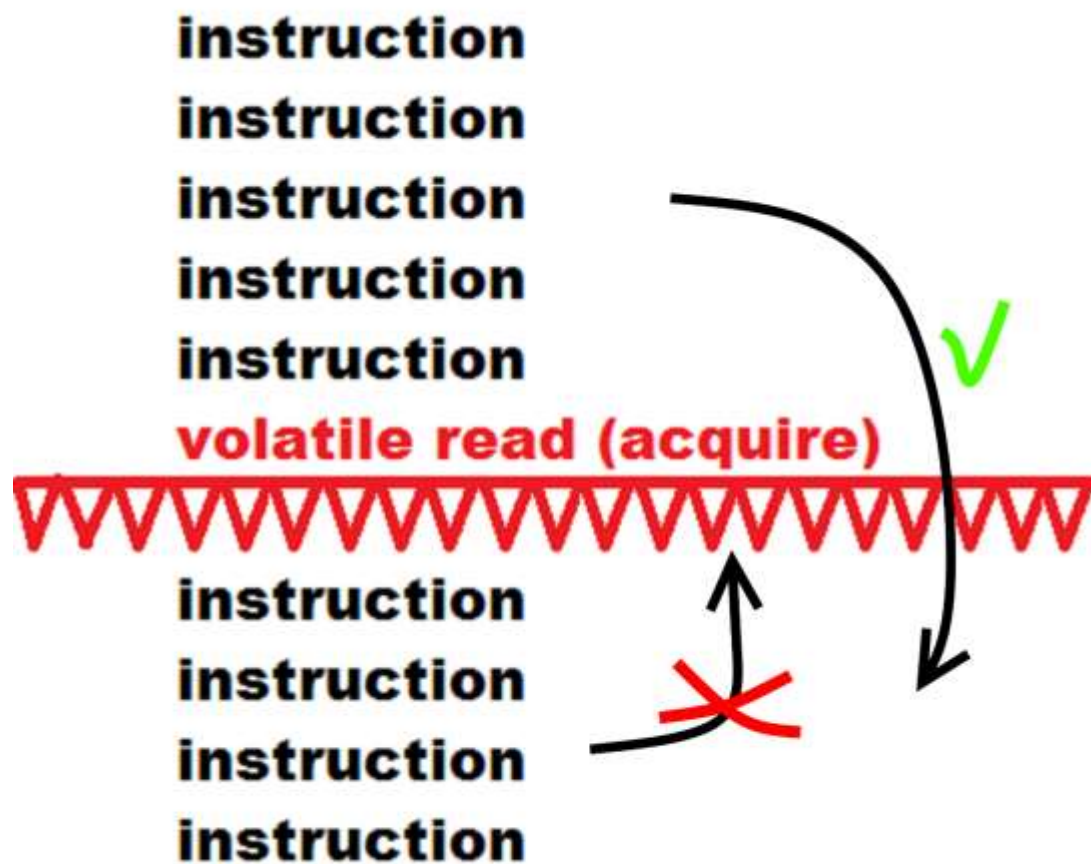
Type	Alpha	ARMv7	PA-RISC	POWER	SPARC RMO	SPARC PSO	SPARC TSO	x86	x86 oostore	AMD64	IA-64	z/Architecture
Loads reordered after loads	Y	Y	Y	Y	Y				Y		Y	
Loads reordered after stores	Y	Y	Y	Y	Y				Y		Y	
Stores reordered after stores	Y	Y	Y	Y	Y	Y			Y		Y	
Stores reordered after loads	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Atomic reordered with loads	Y	Y		Y	Y						Y	
Atomic reordered with stores	Y	Y		Y	Y	Y					Y	
Dependent loads reordered	Y											
Incoherent instruction cache pipeline	Y	Y		Y	Y	Y	Y	Y	Y		Y	

https://en.wikipedia.org/wiki/Memory_ordering

ECMA-335 (I.12.6.4 Optimization)



ECMA-335 (I.12.6.7 Volatile reads and writes)



ECMA-335 (I.12.6.7 Volatile reads and writes)



ECMA-334 (10.5.3 volatile fields)

- Чтение – acquire
- Запись - release

ECMA-335 (I.12.6.5 Locks and threads)

- `System.Threading.Thread.VolatileRead/VolatileWrite/MemoryBarrier`
- `System.Threading.Volatile.Read/Write`
- `System.Threading.Interlocked`
- `System.Threading.Monitor.Enter/Exit`

Безопасен ли этот код?

```
private SomeClass GetOrCreateInstance(KeyClass key, Func<KeyClass, SomeClass> factory)
{
    if (valueInitialized)
        return instance;

    lock (lockObject)
    {
        if (valueInitialized)
            return instance;

        instance = factory(key);
        valueInitialized = true;
        return instance;
    }
}
```


Безопасен ли этот код?

```
private SomeClass GetOrCreateInstance(KeyClass key, Func<KeyClass, SomeClass> factory)
{
    if (valueInitialized)
        return instance;

    lock (lockObject)
    {
        if (valueInitialized)
            return instance;

        instance = factory(key);
        valueInitialized = true;
        return instance;
    }
}
```

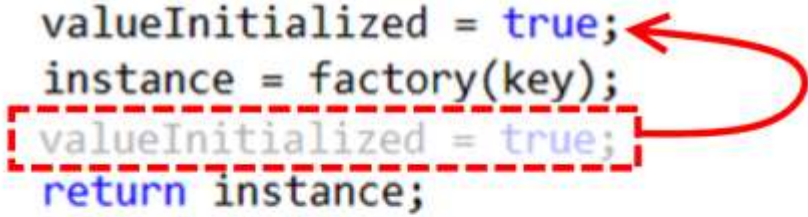
The diagram illustrates a race condition between two threads accessing the `valueInitialized` variable. A horizontal red line at the top represents the execution of the first thread, which performs an **implicit volatile read** of `valueInitialized`. A horizontal blue line at the bottom represents the execution of the second thread, which performs an **implicit volatile write** to `valueInitialized`. Both threads enter the `lock (lockObject)` block. Inside the lock, the second thread checks `if (valueInitialized)` and finds it false, so it proceeds to create the instance and set `valueInitialized = true`. The first thread, which read `valueInitialized` as false before entering the lock, also proceeds to create the instance and set `valueInitialized = true`. This results in two separate instances being created, which is a race condition. The red arrow from the first thread's read to the lock block is marked with a black 'X', and the blue arrow from the second thread's write to the lock block is also marked with a black 'X', indicating that the operations are not atomic and can interfere with each other.

Безопасен ли этот код?

```
private SomeClass GetOrCreateInstance(KeyClass key, Func<KeyClass, SomeClass> factory)
{
    if (valueInitialized)
        return instance;

    lock (lockObject)
    {
        if (valueInitialized)
            return instance;

        valueInitialized = true;
        instance = factory(key);
        valueInitialized = true;
        return instance;
    }
}
```

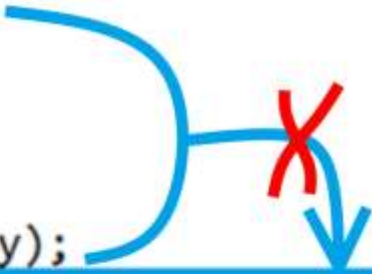


Безопасен ли этот код?

```
private SomeClass GetOrCreateInstance(KeyClass key, Func<KeyClass, SomeClass> factory)
{
    if (valueInitialized)
        return instance;

    lock (lockObject)
    {
        if (valueInitialized)
            return instance;

        instance = factory(key);
        valueInitialized = true;
        return instance;
    }
}
```



Volatile.Write

Вполне безопасно...

```
private volatile SomeClass instance;
private readonly object lockObject = new object();

public SomeClass GetInstance()
{
    // Classic singleton implementation
    if (instance == null)
    {
        lock (lockObject)
        {
            if (instance == null)
            {
                instance = new SomeClass();
            }
        }
    }
    return instance;
}
```

Зачем здесь volatile?

- У меня и без него работает!?
- На StackOverflow говорят, что не надо
- Это в Java надо писать volatile (при этом не работало до 5)!?



3



Note that this is off the cuff, without studying your code closely. I don't *think* Set performs a memory barrier, but I don't see how that's relevant in your code? Seems like more important would be if Wait performs one, which it does. So unless I missed something in the 10 seconds I devoted to looking at your code, I don't believe you need the volatiles.

Edit: Comments are too restrictive. I'm now referring to Matt's edit.

Matt did a good job with his evaluation, but he's missing a detail. First, let's provide some definitions of things thrown around, but not clarified here.

A volatile read reads a value and then invalidates the CPU cache. A volatile write flushes the cache, and then writes the value. A memory barrier flushes the cache and then invalidates it.

Простите, что?

The .NET memory model ensures that all writes are volatile. Reads, by default, are not, unless an explicit VolatileRead is made, or the volatile keyword is specified on the field. Further, interlocked methods force cache coherency, and all of the synchronization concepts (Monitor, ReaderWriterLock, Mutex, Semaphore, AutoResetEvent, ManualResetEvent, etc.) call interlocked methods internally, and thus ensure cache coherency.

Again, all of this is from Jeffrey Richter's book, "CLR via C#".

I said, initially, that I didn't *think* Set performed a memory barrier. However, upon further reflection about what Mr. Richter said, Set would be performing an interlocked operation, and would thus also ensure cache coherency.

I stand by my original assertion that volatile is not needed here.

Edit 2: It looks as if you're building a "future". I'd suggest you look into [PFX](#), rather than rolling your own.

[share](#) [edit](#) [flag](#)

edited Mar 27 '09 at 13:20

answered Mar 25 '09 at 18:09



[wekempf](#)

2,398 ● 9 ● 14

DOTNEXT

Москва
9 декабря

Hardware

Compiler Memory Ordering

- The compiler (CLR) can:
 - Cache (memory into register),
 - Reorder
 - Coalesce writes
- **volatile** keyword disables compiler optimizations.
And that's all!

POSTSHARP

@gfraiteur

И тут он говорит:

«The only thing that volatile keyword does is to disable compiler optimizations.

It doesn't do anything with the CPU. OK?»

Хабрахабр

Публикации

Хабы

Компании

Пользователи

Песочница



Дмитрий Костиков @rumatavz

Пользователь

↑ 34,5

карма

↓

0,0

рейтинг



написать



Профиль

2

Публикации

102

Комментарии

43

Избранное

3

Подписчики

13 октября 2011 в 10:42

Разработка → Барьеры памяти и неблокирующая синхронизация в .NET

из песочницы



Так выглядит модель памяти .NET

Тип перестановки	Перестановка разрешена
Загрузка-загрузка	Да
Загрузка-запись	Да
Запись-загрузка	Да
Запись-запись	Нет

<https://habrahabr.ru/post/130318/>

Зачем здесь volatile!

- x86 и x86-64 имеют весьма строгую модель памяти
- Itanium
- Всё, что не оговорено стандартом, может изменяться в других версиях .NET Framework и/или архитектурах процессоров
- СЮРПРИЗ! У нас есть ARM и Xamarin (будет демо)
- А также Windows 10 IoT Core на Raspberry Pi (но без демо)

Демо.

- Демо про partially constructed object на ARM

Некоторые считают volatile какой-то магией или даже злом.

- Volatile – не магия, а всего лишь ключевое слово!
- Volatile – не зло, а всего лишь ключевое слово!
- Но оно имеет разную семантику в C/C++/C#/Java!
- В C# volatile – это про Acquire-Release семантику и только!

There can only be one!

Вариация на тему алгоритма Петерсона:

```
private volatile bool A;
private volatile bool B;
private volatile bool A_Won;
private volatile bool B_Won;

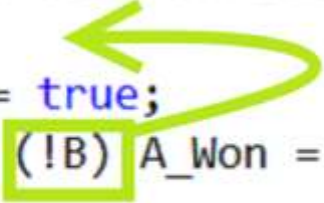
private void ThreadA()
{
    A = true;
    if (!B) A_Won = true;
}

private void ThreadB()
{
    B = true;
    if (!A) B_Won = true;
}
```

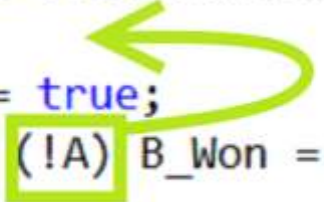
Почему сломано?

```
private volatile bool A;
private volatile bool B;
private volatile bool A_Won;
private volatile bool B_Won;
```

```
private void ThreadA()
{
    A = true;
    if (!B) A_Won = true;
}
```



```
private void ThreadB()
{
    B = true;
    if (!A) B_Won = true;
}
```



Как чинить?

```
private volatile bool A;
private volatile bool B;
private volatile bool A_Won;
private volatile bool B_Won;

private void ThreadA()
{
    A = true;
    Thread.MemoryBarrier();
    if (!B) A_Won = true;
}

private void ThreadB()
{
    B = true;
    Thread.MemoryBarrier();
    if (!A) B_Won = true;
}
```

Вывод

- Стараться использовать более высокоуровневые конструкции
- Читать спецификации
- Писать код в соответствии с ECMA CIL Memory Model

Чего почитать?

- <http://www.albahari.com/threading/>
- [Andrew Tanenbaum “Structured Computer Organization”](#)
- [Jeffrey Richter “CLR via C#”](#)
- <https://habrahabr.ru/company/intel/blog/> (и другие)

Вопросы?



Список литературы:

- **The C# Memory Model in Theory and Practice**
[<https://msdn.microsoft.com/en-us/magazine/jj883956.aspx>]
- **ECMA-335 specification**
[<http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-335.pdf>]
- **ECMA-334 specification**
[%VSINSTALLDIR%\VC#\Specifications\1033\CSharp Language Specification.docx]