

БГТУ, ФИТ, ПОИТ, 3 семестр
Конструирование программного обеспечения

Особенности работы CLR в .NET framework

План лекции:

особенности работы CLR в .NET
шитый код

Основой .NET является общезыковая среда исполнения CLR (Common Language Runtime), в которой могут работать различные языки программирования, такие, например, как C#, F#, J#, Ada, Visual Basic .NET, JScript .NET, C++/CLI и др.

Программа для .NET Framework, написанная на любом поддерживаемом языке программирования, сначала переводится компилятором в единое для .NET промежуточное представление (байт-код) Common Intermediate Language (CIL) (ранее назывался Microsoft Intermediate Language, MSIL, можно просто Intermediate Language, IL) – «высокоуровневый ассемблер» виртуальной машины .NET.

В терминах .NET в результате получается сборка (assembly).

Затем код либо

- исполняется виртуальной машиной Common Language Runtime (CLR);
- транслируется утилитой NGen.exe в исполняемый код для конкретного целевого процессора.

CLR

Общезыковая среда выполнения обычно относится к среде выполнения .NET Framework или среде выполнения .NET 5 (и .NET Core) и более поздних версий.

Среда CLR обеспечивает выделение памяти и управление ею.

Среда CLR является виртуальной машиной, которая не только выполняет приложения, но и компилирует IL-код с помощью JIT-компилятора.

JIT-компилятор.

Аналогично АОТ, этот компилятор преобразует IL в машинный код, который понимает обработчик. В отличие от АОТ, JIT-компиляция происходит по *требованию* и осуществляется на том же компьютере, где должен выполняться код. Так как JIT-компиляция происходит во время выполнения приложения, время компиляции является частью времени выполнения. JIT-компилятор знает фактическое оборудование и может освободить разработчиков от поставки различных реализаций.

Реализация .NET включает в себя:

- одну среду выполнения или несколько (примеры: CLR, CoreRT);
- библиотеку классов, которая реализует версию .NET Standard, а также может содержать дополнительные API-интерфейсы (примеры: BCL для .NET Framework, .NET 5, .NET Core и более поздних версий);
- одна платформа приложений или несколько (примеры: ASP.NET, Windows Forms и WPF включены в платформа .NET Framework и .NET 5+) (необязательно);
- средства разработки (некоторые средства разработки, являются общими для нескольких реализаций) (необязательно).

Примеры реализаций .NET:

- .NET Framework;
- .NET 5 (и .NET Core) и более поздние версии;
- универсальная платформа Windows (UWP);
- Mono (это открытый код кроссплатформенная реализация .NET, для работы в приложениях Xamarin на Android, Mac, iOS, tvOS и watchOS).

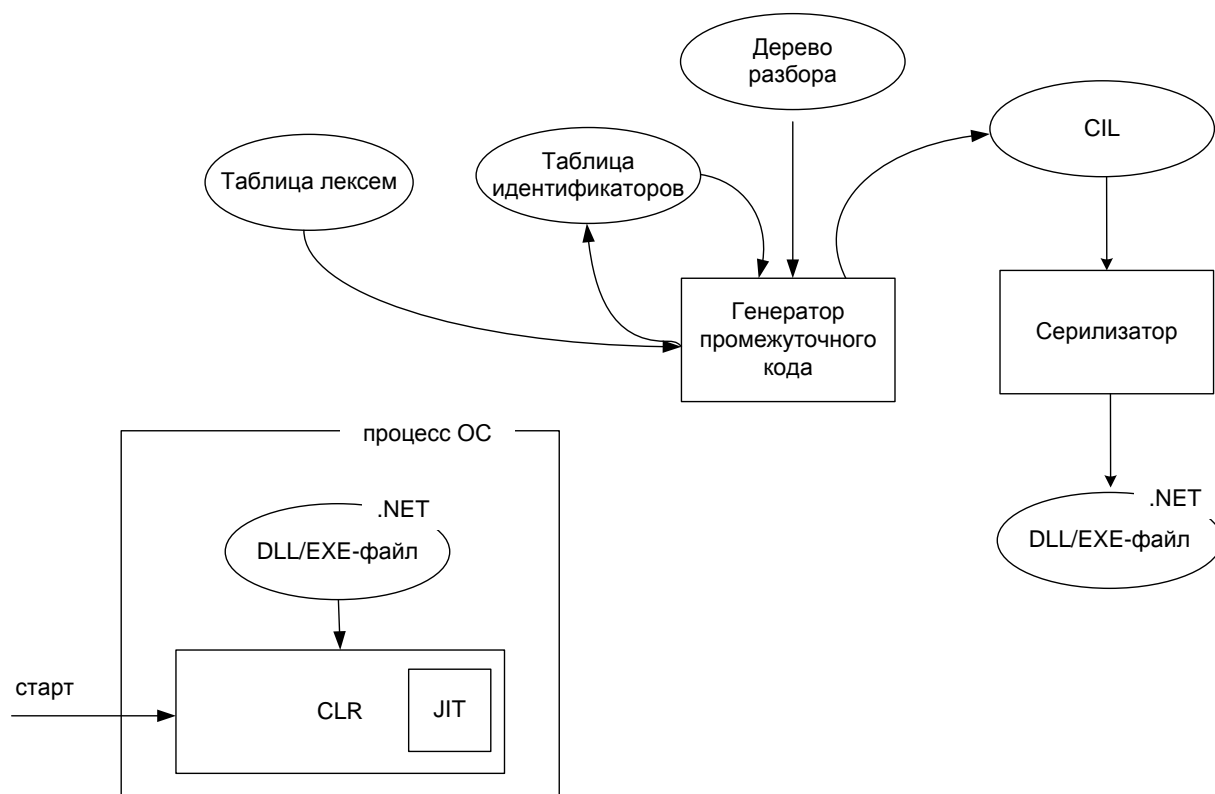
Исторически Mono реализовывала крупный API .NET Framework и эмулировала некоторые из наиболее популярных возможностей в Unix. Иногда она использовалась для запуска приложений .NET, которые применяют эти возможности в Unix.

Архитектура .NET Framework описана и опубликована в спецификации *Common Language Infrastructure (CLI)*, разработанной *Microsoft* и утверждённой *ISO* и *ECMA*.

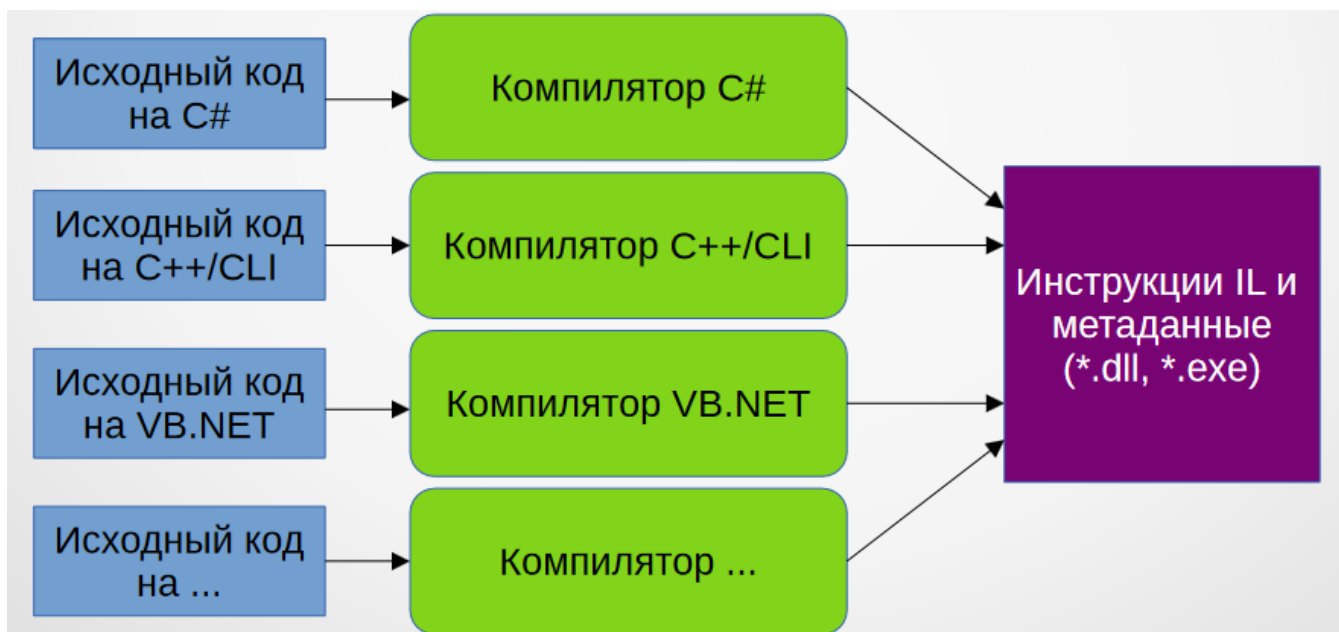
В *CLI* описаны типы данных *.NET*, формат метаданных о структуре программы, система исполнения байт-кода и многое другое.

Объектные классы *.NET*, доступные для всех поддерживаемых языков программирования, содержатся в библиотеке *Framework Class Library (FCL)*. В *FCL* входят классы *Windows Forms*, *ADO.NET*, *ASP.NET*, *Language Integrated Query*, *Windows Presentation Foundation*, *Windows Communication Foundation* и другие. Ядро *FCL* называется *Base Class Library (BCL)*.

1. Генерация кода языка .NET (C#, VB.NET, ...)



1. Программа в .Net, написанная на C#, VB.Net, F#, или каком-либо другом совместимом языке.
2. Этот код компилируется до промежуточного языка (*IL*), который аналогичен байт-коду *Java*, который распространяется на машины конечных пользователей.
3. Конечный пользователь впервые вызывает программу на компьютере с *установленной правильной версией .Net*
4. Компьютер видит, что это *.Net assembly*, а не "raw" машинный код, и передает его компилятору *JIT*
5. Компилятор *JIT компилирует IL в полностью собственный машинный код*.
6. Машинный код сохраняется в памяти на весь срок выполнения этой программы.
7. Вызывается сохраненный машинный код.



Common Language Runtime (CLR) – это имя виртуальной машины, является компонентом в рамках .NET (реализация **Microsoft** стандарта **Common Language Infrastructure (CLI)**, который определяет среду для выполнения программных кодов).

CLR выполняет промежуточный код на **CIL**, который не содержит никаких инструкций, относящихся к оборудованию. **JIT-компилятор** «на лету» преобразует код **CIL** в собственный код, специфичный для операционной системы.

Можно написать приложение непосредственно на **CIL**, языке, который выглядит как ассемблер.

CLR требуется для запуска промежуточного кода. Он основан на **CTS** (**Common Type System**) и **CLS** (**Common Language Specification**). Он предоставляет множество функций, таких как сборщик мусора, **BCL** (библиотека базовых классов) и система безопасности. Код поступает на вход **CLR** и скомпилируется **JIT**-компилятором (точно в срок) на языке целевой машины. целевая платформа

Коллекция **API**-интерфейсов, которую использует приложение или библиотека **.NET**.

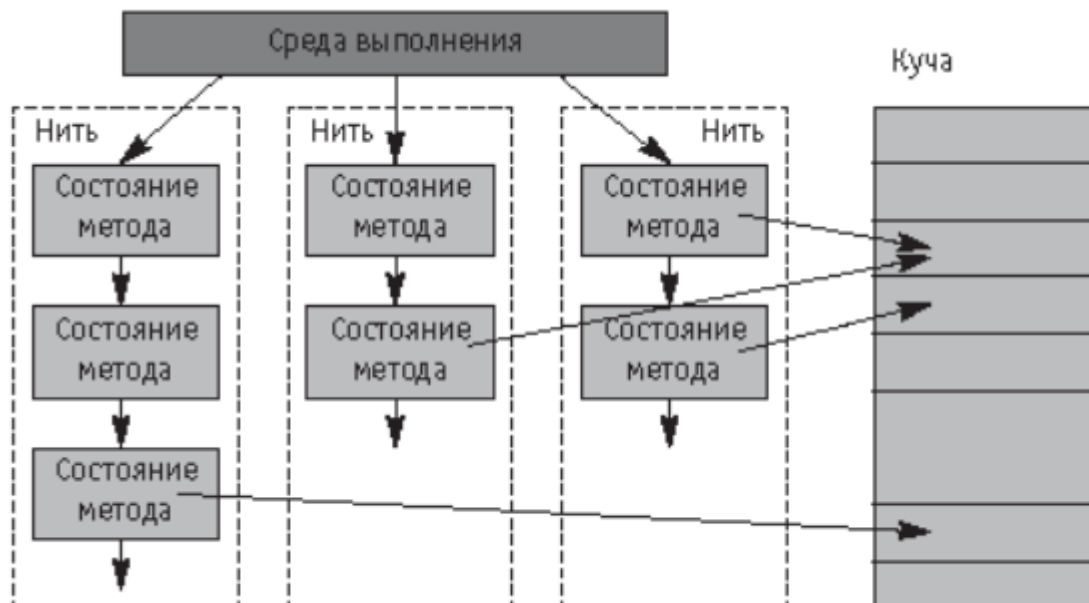
CLR (виртуальная машина) состоит из следующих четырех частей:

- система общих типов (**CTS**);
- спецификация общего языка (**CLS**);
- метаданные;
- виртуальная система исполнения (**VES**).

Виртуальная система выполнения (**Virtual Execution System – VES**) представляет собой абстрактную виртуальную машину для выполнения управляемого кода.

Ни одна из реализаций виртуальной машины стандарта **CLI** не содержит интерпретатора **CIL**-кода (используется **JIT**-компилятор, транслирующий инструкции **CIL** в команды процессора).

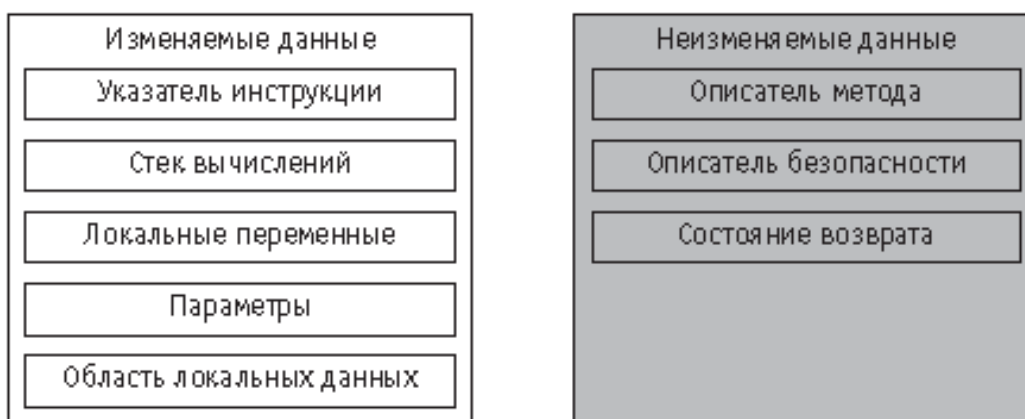
Схема состояния виртуальной машины:



Виртуальная машина может выполнять сразу несколько нитей (threads).

Состояние метода

Схема состояния метода:



Элементы состояния метода, входящие в группу изменяемых данных:

- Указатель инструкции (Instruction Pointer).

Содержит адрес следующей инструкции в теле метода, которая будет выполнена системой выполнения. (Когда мы говорим, что указатель инструкции относится к изменяемым данным, мы имеем в виду, что его значение изменяется при переходе от инструкции к инструкции)

- Стек вычислений (Evaluation Stack).

Виртуальная система выполнения работает по принципу стекового процессора. Это означает, что операнды инструкций, а также возвращаемые инструкциями значения хранятся в специальной области памяти, а именно на стеке вычислений.

Каждое состояние метода имеет собственный стек вычислений, содержимое которого сохраняется при вызове методов (то есть, если наш метод вызывает другой метод, то по завершении работы вызванного метода содержимое стека никуда не денется).

- **Локальные переменные (Local Variable Array).**

Для хранения локальных переменных в состоянии метода предусмотрена отдельная область памяти, состоящая из так называемых слотов (slots). Каждой локальной переменной соответствует свой слот. Значения локальных переменных сохраняются при вызове методов аналогично содержимому стека вычислений.

- **Параметры (Argument Array).**

Фактические параметры, переданные методу, записываются в специальную область памяти, которая организована так же, как и область локальных переменных.

- **Область локальных данных (Local Memory Pool).**

В языке CIL предусмотрена инструкция `localloc`, которая позволяет динамически размещать объекты в области памяти, локальной для метода. Объекты в этой области живут до тех пор, пока метод не завершится.

Обратите внимание, что стек вычислений, локальные переменные и параметры, а также локальные данные метода представляют собой логически отдельные области памяти. Каждая конкретная реализация CLI самостоятельно решает вопрос, где размещать эти области.

Неизменяемые данные:

- **Описатель метода (MethodInfo handle).**

Содержит сигнатуру метода, в которую входят количество и типы формальных параметров, а также тип возвращаемого значения. Кроме этого, описатель метода включает в себя информацию о количестве и типах локальных переменных и об обработчиках исключений.

Описатель метода доступен из кода метода, но в основном он используется системой выполнения при сборке мусора и обработке исключений.

- **Описатель безопасности (Security Descriptor).**

Используется системой безопасности CLI и недоступен из кода метода.

- **Состояние возврата (Return State Handle).**

Служит для организации списка состояний методов внутри системы выполнения и недоступно из кода метода. Фактически представляет собой указатель на состояние метода, из тела которого был вызван текущий метод.

Стек вычислений

Стек вычислений в *VES* состоит из слотов. При этом глубина стека (максимальное количество слотов) **всегда ограничена** и задается статически в заголовке метода. Решение ограничить глубину стека было принято разработчиками спецификации *CLI* для того, чтобы облегчить создание *JIT*-компиляторов.

На входе метода стек вычислений всегда пуст. Затем он используется для передачи операндов инструкциям CIL, для передачи фактических параметров

вызываемым методам, а также для получения результатов выполнения инструкций и вызываемых методов. Если метод возвращает какое-то значение, то оно кладется на стек вычислений перед завершением метода.

Каждый слот стека вычислений может содержать ровно одно значение одного из следующих типов:

- ***int64*** – 8-байтовое целое со знаком;
- ***int32*** – 4-байтовое целое со знаком;
- ***native int*** – знаковое целое, разрядность которого зависит от аппаратной платформы (может быть 4 или 8 байт);
- ***F*** – число с плавающей точкой, разрядность которого зависит от аппаратной платформы (не может быть меньше 8 байт);
- ***&*** – управляемый указатель;
 - – объектная ссылка;
- пользовательский тип-значение.

Слоты стека вычислений могут иметь различный размер в зависимости от типов записанных в них значений.

Также мы можем видеть, что допустимые типы значений для стека вычислений не совпадают с общей системой типов CTS. Например, в CTS существуют целые типы разрядности 1 и 2 байта, которые не могут содержаться на стеке вычислений. И наоборот, тип *F* стека вычислений не имеет аналога в CTS. Кроме того, для стека вычислений все управляемые указатели и объектные ссылки отображаются в два типа: *&* и *O* соответственно.

Формат метаданных.

Синтаксис и мнемоника языка CIL описываются стандартом «ЕСМА-335». Спецификация CIL является составной частью более общей спецификации — спецификации CLI (англ. common language infrastructure).

Преимущества:

1) Вся платформа .NET основана на единой объектно-ориентированной модели. Что это значит? Все сервисы, интерфейсы и объекты, которые платформа предоставляет разработчику объединены в единую иерархию классов, сгруппированную удобно и интуитивно понятно.

2) Приложение, написанное на любом .NET-совместимом языке является межплатформенным.

3) В состав платформы .NET входит т.н. «сборщик мусора», который освобождает ресурсы. Таким образом, приложения защищены от утечки памяти и от необходимости освобождать ресурсы. Это делает программирование более легким и более безопасным.

4) Приложения .NET используют метаданные, что позволяет им *не пользоваться* системным реестром Windows.

5) Любое .NET приложение является автономным, в том смысле, что не зависит от других программ, в частности от ОС. Установка приложения, написанного на одном из .NET языках может быть произведена обычным копированием файлов.

6) Приложения .NET используют безопасные типы, что повышает их надежность, совместимость и межплатформенность.

7) Приложения, написанные на разных языках могут легко взаимодействовать. Например, серверная часть может быть написана на C#, а клиентская на другом.

9) .NET приложения могут быть сертифицированы на безопасность. Это является особенностью промежуточного кода, в который преобразуются все .NET приложения.

10) Абсолютно все ошибки обрабатываются механизмом исключительных ситуаций. Это позволяет избежать разногласия, которые иногда возникали при программировании под Win32.

11) Повторное использование кода стало еще удобнее. Это связано с тем, что промежуточный язык MSIL не зависит от языка программирования. Например, вы можете написать программу на C#, а патч к ней писать уже, скажем, на J#.

Отличия от виртуальной машиной *Java*.

Программы на *Java* транслируются в байт-код, выполняемый виртуальной машиной *Java (JVM)* — программой, обрабатывающей байтовый код и передающей инструкции оборудованию как интерпретатор. Байт-код, в отличие от исходного текста, обрабатывается значительно быстрее.

Достоинство подобного способа выполнения программ — в полной независимости байт-кода от *операционной системы и оборудования*, что позволяет выполнять *Java*-приложения на любом устройстве, для которого существует соответствующая виртуальная машина. Другой важной особенностью технологии *Java* является гибкая система безопасности благодаря тому, что исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером) вызывают немедленное прерывание.

Если сравнить CLI с ее ближайшим конкурентом – платформой Java.

Можно прийти к выводу, что VES является значительно более абстрактной моделью, чем виртуальная машина Java (Java Virtual Machine – JVM). Причина такого отличия кроется в том, что изначально Java была ориентирована на реализацию в бытовых приборах. При этом, естественно, подразумевалось, что байт-код Java будет непосредственно выполняться специальными процессорами, и поэтому JVM является *фактически спецификацией такого процессора*.

Аппаратная реализация *VES* никогда даже не предполагалась, и это позволило избежать при составлении ее спецификации ненужных деталей, дав тем самым каждой реализации *CLI* большую свободу выбора наиболее оптимальной стратегии выполнения *CIL*-кода.

Существует три типа *JIT*:

- *Pre-JIT*: весь код компилируется напрямую;
- *Econo-JIT*: код компилируется по частям, и при необходимости освобождается память;
- Нормальный-*JIT* (по умолчанию): код компилируется только при необходимости, но затем кэшируется для повторного использования.

Простое приложение ASP.NET.

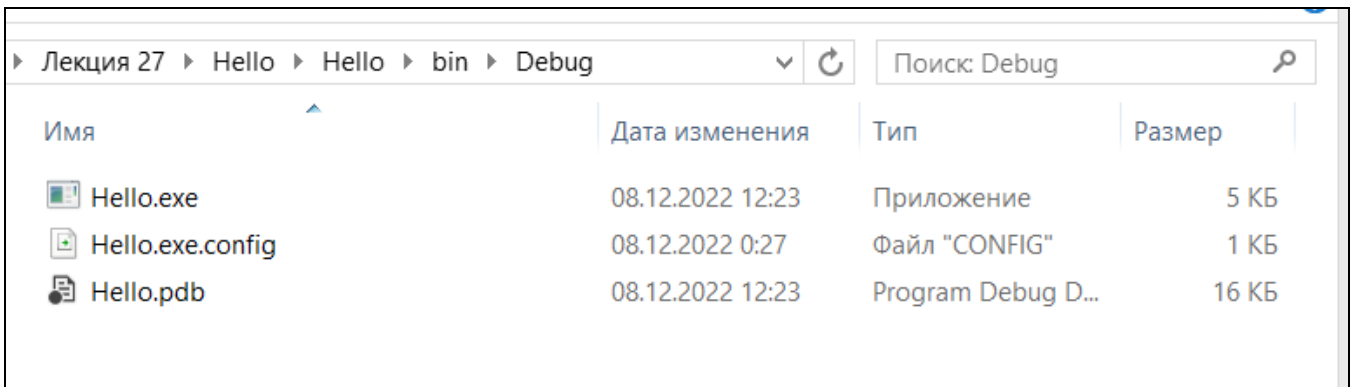
Пример кода




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Hello
{
    class Program
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
            Hello2();
        }

        static void Hello2()
        {
            Console.WriteLine("Hello Students!");
        }
    }
}
```

В папке появятся 3 файла, первый из них мы откроем с помощью утилиты **ILDasm (IL Disassembler)**, который устанавливается вместе с **Visual Studio** (позволяет просматривать код CIL).



Имя	Дата изменения	Тип	Размер
 Hello.exe	08.12.2022 12:23	Приложение	5 КБ
 Hello.exe.config	08.12.2022 0:27	Файл "CONFIG"	1 КБ
 Hello.pdb	08.12.2022 12:23	Program Debug D...	16 КБ

```
Windows PowerShell

*****
** Visual Studio 2019 Developer Command Prompt v16.3.1
** Copyright (c) 2019 Microsoft Corporation
*****

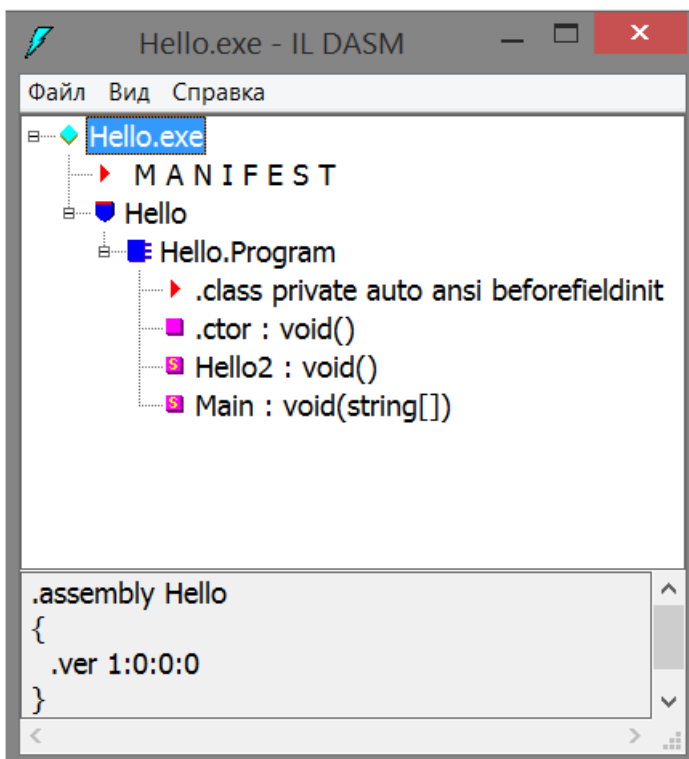
D:\Adel\Кафедра\КПО\Лекции\Осенний семестр\Лекция 27\Hello>cd Hello\bin\Debug

D:\Adel\Кафедра\КПО\Лекции\Осенний семестр\Лекция 27\Hello\Hello\bin\Debug>Hello.exe
Hello World!
Hello Students!

D:\Adel\Кафедра\КПО\Лекции\Осенний семестр\Лекция 27\Hello\Hello\bin\Debug>ildasm Hello.exe

D:\Adel\Кафедра\КПО\Лекции\Осенний семестр\Лекция 27\Hello\Hello\bin\Debug>
```

Манифест сборки и имя проекта:



В “Main : void(string[])” IL-код метода Main, который во время выполнения программы JIT-компилятор генерирует в машинный код.

Hello.Program::.class private auto ansi beforefieldinit

Найти Найти далее

```
.class private auto ansi beforefieldinit Hello.Program
    extends [mscorlib]System.Object
{
} // end of class Hello.Program
```

Hello.Program::.ctor : void()

Найти Найти далее

```
.method public hidebysig specialname rtspecialname
    instance void .ctor() cil managed
{
    // Размер кода:      8 (0x8)
    .maxstack 8
    IL_0000: ldarg.0
    IL_0001: call        instance void [mscorlib]System.Object::.ctor()
    IL_0006: nop
    IL_0007: ret
} // end of method Program::.ctor
```

Hello.Program::Hello2 : void()

Найти Найти далее

```
.method private hidebysig static void Hello2() cil managed
{
    // Размер кода:      13 (0xd)
    .maxstack 8
    IL_0000: nop
    IL_0001: ldstr        "Hello Students!"
    IL_0006: call        void [mscorlib]System.Console::WriteLine(string)
    IL_000b: nop
    IL_000c: ret
} // end of method Program::Hello2
```

Hello.Program::Main : void(string[])

Найти Найти далее

```
.method public hidebysig static void Main(string[] args) cil managed
{
    .entrypoint
    // Размер кода:      19 (0x13)
    .maxstack 8
    IL_0000: nop
    IL_0001: ldstr        "Hello World!"
    IL_0006: call        void [mscorlib]System.Console::WriteLine(string)
    IL_000b: nop
    IL_000c: call        void Hello.Program::Hello2()
    IL_0011: nop
    IL_0012: ret
} // end of method Program::Main
```

Видно, что этот код во время компиляции «переведен» на IL язык с соответствующим определением метаданных и все это станет одним управляемым модулем, который войдет в управляемую сборку.

Все поддерживаемые в CIL **коды операций** (которых немало) могут быть разделены на три основных категории:

- коды операций, позволяющие управлять выполнением программы;
- коды операций, позволяющие вычислять выражения;
- коды операций, позволяющие получать доступ к значениям в памяти (через параметры, локальные переменные и т.д.).

В таблице приведены некоторые коды операций

Коды операций	Описание
add, sub, mul, div, rem	Позволяют выполнять сложение, вычитание, умножение и деление двух значений (rem возвращает остаток от деления)
and, or, not, xor	Позволяют выполнять соответствующие побитовые операции над двумя значениями
seq, cgt, clt	Позволяют сравнивать два значения в стеке различными способами:
ret	Применяется для выхода из метода и (если необходимо) возврата значения вызывающему коду
beq, bgt, ble, bit, switch	Применяются (вместе с другими похожими кодами операций) для управления логикой ветвления внутри метода (переходы по равно, по больше, ...)
call	Применяется для вызова члена определенного типа

Все связанные с ветвлением коды операций требуют указания в CIL-коде метки, к которой должен осуществляться переход в случае, если результат проверки оказывается истинным

Работа со стеком:

Код операций	Описание
ldarg	Позволяет загружать в стек аргумент метода. Помимо основного варианта ldarg (который работает с индексом, представляющим аргумент), существует множество других вариантов. Например, есть варианты ldarg, которые работают с числовым суффиксом (ldarg_0) и позволяют жестко кодировать загружаемый аргумент. Также есть варианты, позволяющие жестко кодировать как только один тип данных, к которому относится аргумент, с помощью константной нотации CIL (например, ldarg_I4 для int32), так и тип данных и значение

	вместе (например, <code>ldarg_I4_5</code> , позволяющий загружать <code>int32</code> со значением 5)
<code>ldc</code>	Позволяет загружать в стек значение константы
<code>ldfld</code>	Позволяет загружать в стек значение поля уровня экземпляра
<code>ldloc</code>	Позволяет загружать в стек значение локальной переменной
<code>ldobj</code>	Позволяет получать все значения размещаемого в куче объекта и помещать их в стек
<code>ldstr</code>	Позволяет загружать в стек строковое значение
<code>pop</code>	Позволяет удалять значение, которое в текущий момент находится на верхушке стека вычислений, но не сохранять его
<code>starg</code>	Позволяет сохранять самое верхнее значение из стека в аргументе метода с определенным индексом
<code>stloc</code>	Позволяет извлекать текущее значение из верхушки стека вычислений и сохранять его в списке локальных переменных с определенным индексом
<code>stobj</code>	Позволяет копировать значение определенного типа из стека вычислений в память по определенному адресу
<code>stsfld</code>	Позволяет заменять значение статического поля значением из стека вычислений

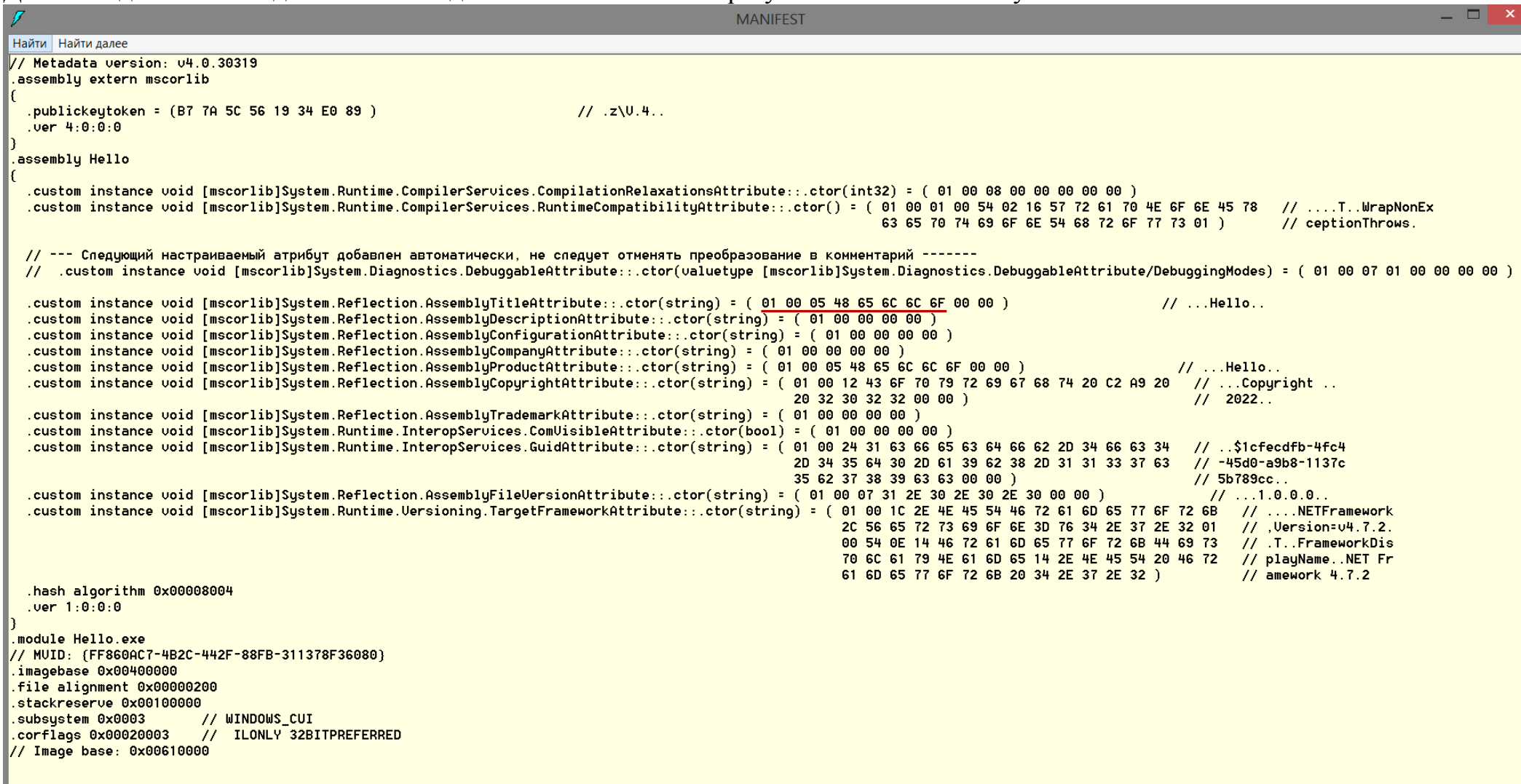
Директива `.maxstack`

Эта директива позволяет указать максимальное количество переменных, которое может помещаться в стек в любой момент во время выполнения метода. Директива имеет значение ***по умолчанию*** 8, подходящее для подавляющего большинства создаваемых методов. При желании можно вручную вычислять количество локальных переменных в стеке и указывать его явно.

Манифест

Так выглядят метаданные для примера Hello (таблицы метаданных преобразованы в понятный вид с помощью дизассемблера `ILDasm.exe`).

Для исследования метаданных и IL-кода можно использовать результаты выполнения утилиты ILDasm.exe:



```
// Metadata version: v4.0.30319
.assembly extern mscorlib
{
    .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )           // .z\U.4..
    .ver 4:0:0:0
}
.assembly Hello
{
    .custom instance void [mscorlib]System.Runtime.CompilerServices.CompilationRelaxationsAttribute::.ctor(int32) = ( 01 00 08 00 00 00 00 00 )
    .custom instance void [mscorlib]System.Runtime.CompilerServices.RuntimeCompatibilityAttribute::.ctor() = ( 01 00 01 00 54 02 16 57 72 61 70 4E 6F 6E 45 78 // ....T..WrapNonEx
63 65 70 74 69 6F 6E 54 68 72 6F 77 73 01 )           // ceptionThrows.

    // --- Следующий настраиваемый атрибут добавлен автоматически, не следует отменять преобразование в комментарий -----
    // .custom instance void [mscorlib]System.Diagnostics.DebuggableAttribute::.ctor(valuetype [mscorlib]System.Diagnostics.DebuggableAttribute/DebuggingModes) = ( 01 00 07 01 00 00 00 00 )

    .custom instance void [mscorlib]System.Reflection.AssemblyTitleAttribute::.ctor(string) = ( 01 00 05 48 65 6C 6C 6F 00 00 )           // ...Hello..
    .custom instance void [mscorlib]System.Reflection.AssemblyDescriptionAttribute::.ctor(string) = ( 01 00 00 00 00 00 )
    .custom instance void [mscorlib]System.Reflection.AssemblyConfigurationAttribute::.ctor(string) = ( 01 00 00 00 00 00 )
    .custom instance void [mscorlib]System.Reflection.AssemblyCompanyAttribute::.ctor(string) = ( 01 00 00 00 00 00 )
    .custom instance void [mscorlib]System.Reflection.AssemblyProductAttribute::.ctor(string) = ( 01 00 05 48 65 6C 6C 6F 00 00 )           // ...Hello..
    .custom instance void [mscorlib]System.Reflection.AssemblyCopyrightAttribute::.ctor(string) = ( 01 00 12 43 6F 70 79 72 69 67 68 74 20 C2 A9 20 // ...Copyright ..
20 32 30 32 32 00 00 )           // 2022..
    .custom instance void [mscorlib]System.Reflection.AssemblyTrademarkAttribute::.ctor(string) = ( 01 00 00 00 00 00 )
    .custom instance void [mscorlib]System.Runtime.InteropServices.ComVisibleAttribute::.ctor(bool) = ( 01 00 00 00 00 00 )
    .custom instance void [mscorlib]System.Runtime.InteropServices.GuidAttribute::.ctor(string) = ( 01 00 24 31 63 66 65 63 64 66 62 2D 34 66 63 34 // ..$1cfecdfb-4fc4
2D 34 35 64 30 2D 61 39 62 38 2D 31 31 33 37 63 // -45d0-a9b8-1137c
35 62 37 38 39 63 63 00 00 )           // 5b789cc..
    .custom instance void [mscorlib]System.Reflection.AssemblyFileVersionAttribute::.ctor(string) = ( 01 00 07 31 2E 30 2E 30 2E 30 00 00 )           // ...1.0.0.0..
    .custom instance void [mscorlib]System.Runtime.Versioning.TargetFrameworkAttribute::.ctor(string) = ( 01 00 1C 2E 4E 45 54 46 72 61 6D 65 77 6F 72 6B // ....NETFramework
2C 56 65 72 73 69 6F 6E 3D 76 34 2E 37 2E 32 01 // ,Version=v4.7.2.
00 54 0E 14 46 72 61 6D 65 77 6F 72 6B 44 69 73 // .T..FrameworkDis
70 6C 61 79 4E 61 6D 65 14 2E 4E 45 54 20 46 72 // playName...NET Fr
61 6D 65 77 6F 72 6B 20 34 2E 37 2E 32 )           // amework 4.7.2

    .hash algorithm 0x00008004
    .ver 1:0:0:0
}
.module Hello.exe
// MUID: {FF860AC7-4B2C-442F-88FB-311378F36080}
.imagebase 0x00400000
.file alignment 0x00000200
.stackreserve 0x00100000
.subsystem 0x0003 // WINDOWS_CUI
.corflags 0x00020003 // ILONLY 32BITPREFERRED
// Image base: 0x00610000
```

Назначение некоторых атрибутов:

- AssemblyCompany: название компании
- AssemblyConfiguration: конфигурация сборки (Retail или Debug)
- AssemblyCopyright: авторское право на программу
- AssemblyDefaultAlias: псевдоним по умолчанию, используемый при ссылке на данную сборку из других сборок
- AssemblyDescription: краткое описание сборки
- AssemblyProduct: информация о продукте
- AssemblyTitle: название сборки как информационного продукта
- AssemblyTrademark: сведения о торговой марке
- AssemblyCulture: задает язык и региональные параметры, поддерживаемые сборкой (например, установка русской культуры: `[assembly:AssemblyCultureAttribute("ru")]`)
- AssemblyInformationalVersion: полная версия сборки
- AssemblyVersion: версия сборки
- AssemblyFileVersion: номер версии файла Win32. По умолчанию совпадает с версией сборки.

В начале данных манифеста определение публичного токена *mscorlib* которое будет использовано для доступа к этой сборке в GAC.

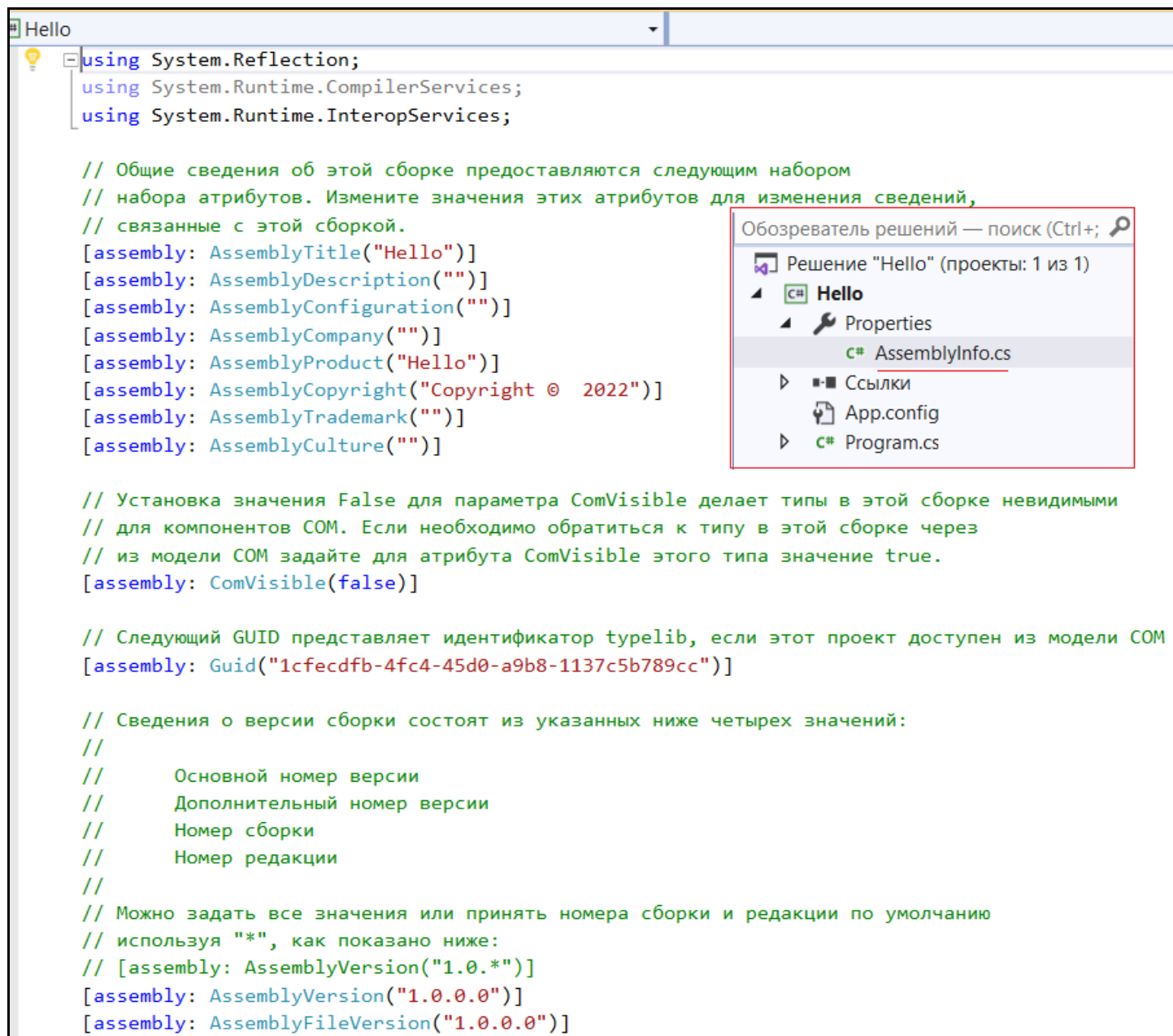
Ключевым компонентом сборки является ее манифест. Если у сборки отсутствует манифест, то заключенный в ней код MSIL выполняться не будет. Манифест может находиться в одном файле с исполняемым кодом сборки, а может размещаться и в отдельном файле.

Манифест хранит следующие данные:

- Имя сборки
- Номер версии: основной и дополнительный номера. Используется для управления версиями
- Язык и региональные параметры: информация о языке и региональных параметрах, которые поддерживает сборка
- Информация о строгом имени: открытый ключ издателя
- Список всех файлов сборки: хэш и имя каждого из входящих в сборку файлов
- Список ссылок на другие сборки, которые использует текущая сборка
- Список ссылок на типы, используемые сборкой

Манифест позволяет системе определить все файлы, входящие в сборку, сопоставить ссылки на типы, ресурсы, сборки с их файлами, управлять контролем версий.

По умолчанию Visual Studio при создании проекта добавляет файл AssemblyInfo.cs, который можно найти в узле Properties «Обозревателя решений»:



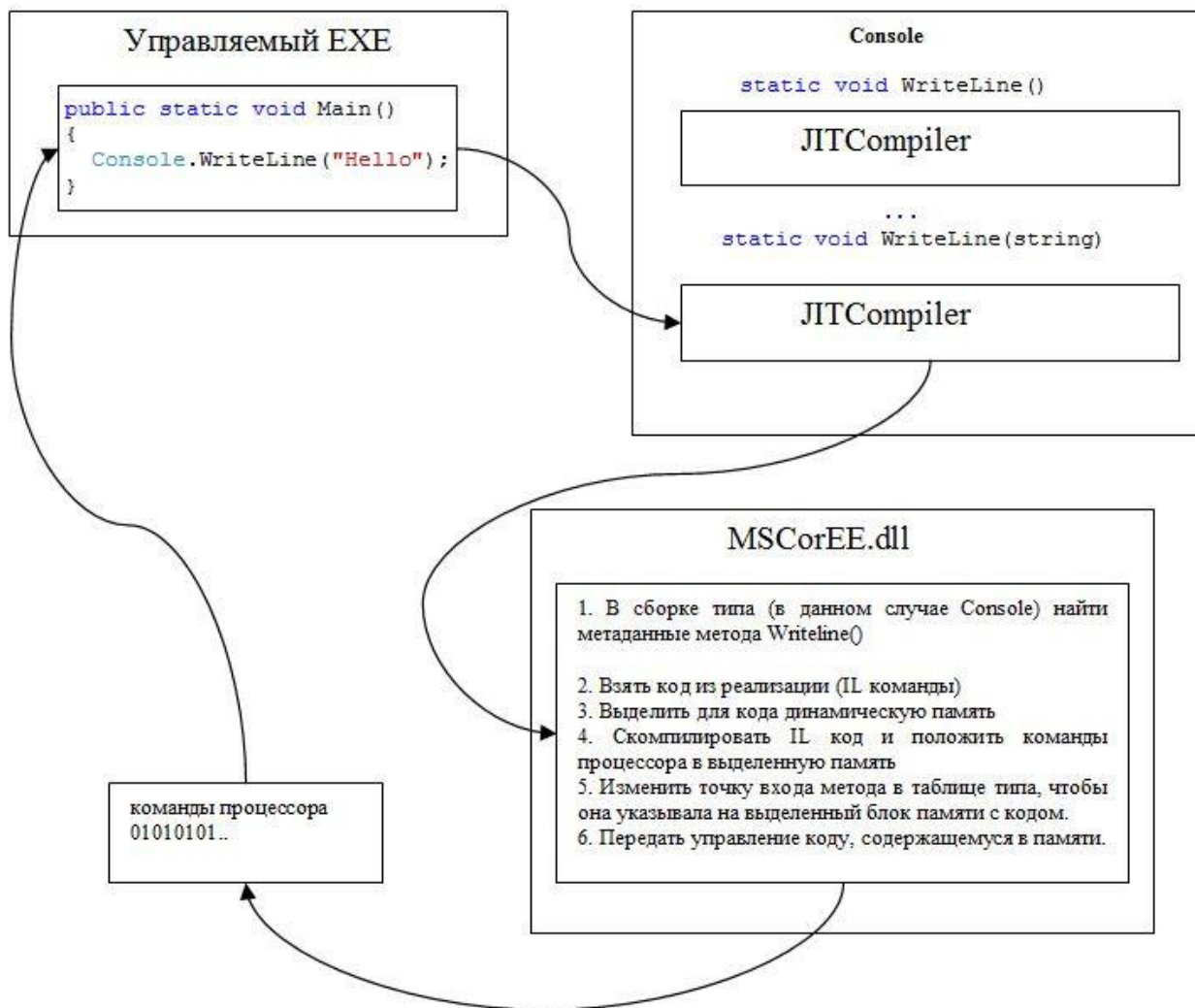
```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

// Общие сведения об этой сборке предоставляются следующим набором
// набора атрибутов. Измените значения этих атрибутов для изменения сведений,
// связанные с этой сборкой.
[assembly: AssemblyTitle("Hello")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("Hello")]
[assembly: AssemblyCopyright("Copyright © 2022")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Установка значения False для параметра ComVisible делает типы в этой сборке невидимыми
// для компонентов COM. Если необходимо обратиться к типу в этой сборке через
// из модели COM задайте для атрибута ComVisible этого типа значение true.
[assembly: ComVisible(false)]

// Следующий GUID представляет идентификатор typelib, если этот проект доступен из модели COM
[assembly: Guid("1cfecdfe-4fc4-45d0-a9b8-1137c5b789cc")]

// Сведения о версии сборки состоят из указанных ниже четырех значений:
//
//      Основной номер версии
//      Дополнительный номер версии
//      Номер сборки
//      Номер редакции
//
// Можно задать все значения или принять номера сборки и редакции по умолчанию
// используя "*", как показано ниже:
// [assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```



Компилятор АОТ.

Аналогично JIT, этот компилятор также преобразует IL в машинный код. В отличие от JIT-компиляции АОТ-компиляция происходит до выполнения приложения и обычно осуществляется на другом компьютере. Так как цепочки средств АОТ не компилируются во время выполнения, им не нужно сокращать время, затрачиваемое на компиляцию. Это означает, что они могут тратить больше времени на оптимизацию. Так как контекстом АОТ является все приложение, АОТ-компилятор также выполняет межмодульное связывание и анализ всей программы. Это означает, что соблюдаются все ссылки и создается один исполняемый файл.

Сборка может включать разные типы, например интерфейсы, классы, структуры, перечисления и делегаты. Сборки в папке bin проекта иногда называют двоичными файлами.

С помощью команды:

```
dumpbin /all Hello.exe > result.txt
```

можно получить дампы памяти и сохранить результат в текстовом файле.

```
·00402060:·00·06·00·2A·36·00·72·1B·00·00·70·28·0F·00·00·0A·...*6.r...p(....  
·00402070:·00·2A·22·02·28·10·00·00·0A·00·2A·00·42·53·4A·42·...*".....*BSJB  
·00402080:·01·00·01·00·00·00·00·00·0C·00·00·00·76·34·2E·30·.....v4.0
```

Начало блока определения метаданных обозначено байтами 42 53 4A 42 (BSJB), которые являются первыми буквами имен разработчиков реализовавших метаданные во фреймворке .NET. По дампу можно исследовать метаданные и IL-код.

Содержимое файла result.txt:

Microsoft (R) COFF/PE Dumper Version 14.23.28105.4
Copyright (C) Microsoft Corporation. All rights reserved.

Dump of file Hello.exe

PE signature found

File Type: EXECUTABLE IMAGE

FILE HEADER VALUES

- 14C machine (x86)
- 3 number of sections
- E2CBF08A time date stamp
- 0 file pointer to symbol table
- 0 number of symbols
- E0 size of optional header
- 22 characteristics
 - Executable
 - Application can handle large (>2GB) addresses

OPTIONAL HEADER VALUES

- 10B magic # (PE32)
- 48.00 linker version
- 800 size of code
- 800 size of initialized data
- 0 size of uninitialized data
- 2792 entry point (00402792)
- 2000 base of code
- 4000 base of data
- 400000 image base (00400000 to 00407FFF)
- 2000 section alignment
- 200 file alignment
- 4.00 operating system version
- 0.00 image version
- 6.00 subsystem version
 - 0 Win32 version
- 8000 size of image
- 200 size of headers
- 0 checksum
- 3 subsystem (Windows CUI)
- 8560 DLL characteristics
 - High Entropy Virtual Addresses

Dynamic base
 NX compatible
 No structured exception handler
 Terminal Server Aware
 100000 size of stack reserve
 1000 size of stack commit
 100000 size of heap reserve
 1000 size of heap commit
 0 loader flags
 10 number of directories
 0 [0] RVA [size] of Export Directory
 273D [4F] RVA [size] of Import Directory
 4000 [58C] RVA [size] of Resource Directory
 0 [0] RVA [size] of Exception Directory
 0 [0] RVA [size] of Certificates Directory
 6000 [C] RVA [size] of Base Relocation Directory
 2674 [38] RVA [size] of Debug Directory
 0 [0] RVA [size] of Architecture Directory
 0 [0] RVA [size] of Global Pointer Directory
 0 [0] RVA [size] of Thread Storage Directory
 0 [0] RVA [size] of Load Configuration Directory
 0 [0] RVA [size] of Bound Import Directory
 2000 [8] RVA [size] of Import Address Table Directory
 0 [0] RVA [size] of Delay Import Directory
 2008 [48] RVA [size] of COM Descriptor Directory
 0 [0] RVA [size] of Reserved Directory

SECTION HEADER #1

.text name
 798 virtual size
 2000 virtual address (00402000 to 00402797)
 800 size of raw data
 200 file pointer to raw data (00000200 to 000009FF)
 0 file pointer to relocation table
 0 file pointer to line numbers
 0 number of relocations
 0 number of line numbers
 60000020 flags
 Code
 Execute Read

RAW DATA #1

00402000: 71 27 00 00 00 00 00 00 48 00 00 00 02 00 05 00 q'.....H.....
 00402010: 7C 20 00 00 F8 05 00 00 03 00 02 00 01 00 00 06 |..o.....

```

00402020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00402030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00402040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00402050: 4E 00 72 01 00 00 70 28 0F 00 00 0A 00 28 02 00 N.r..p(....(..
00402060: 00 06 00 2A 36 00 72 1B 00 00 70 28 0F 00 00 0A ...*6.r..p(....
00402070: 00 2A 22 02 28 10 00 00 0A 00 2A 00 42 53 4A 42 .*".(.....*BSJB
00402080: 01 00 01 00 00 00 00 00 0C 00 00 00 76 34 2E 30 .....v4.0
00402090: 2E 33 30 33 31 39 00 00 00 00 05 00 6C 00 00 00 .30319.....l...
004020A0: D8 01 00 00 23 7E 00 00 44 02 00 00 54 02 00 00 O..#~.D...T...
004020B0: 23 53 74 72 69 6E 67 73 00 00 00 00 98 04 00 00 #Strings.....
004020C0: 3C 00 00 00 23 55 53 00 D4 04 00 00 10 00 00 00 <...#US.O.....
004020D0: 23 47 55 49 44 00 00 00 E4 04 00 00 14 01 00 00 #GUID...a.....
004020E0: 23 42 6C 6F 62 00 00 00 00 00 00 00 02 00 00 01 #Blob.....
004020F0: 47 15 00 00 09 00 00 00 00 FA 01 33 00 16 00 00 G.....u.3....
00402100: 01 00 00 00 11 00 00 00 02 00 00 00 03 00 00 00 .....
00402110: 01 00 00 00 10 00 00 00 0E 00 00 00 01 00 00 00 .....
00402120: 01 00 00 00 00 00 91 01 01 00 00 00 00 00 06 00 .....
00402130: 06 01 19 02 06 00 73 01 19 02 06 00 3A 00 E7 01 .....s.....:c.
00402140: 0F 00 39 02 00 00 06 00 62 00 C9 01 06 00 E9 00 ..9....b.E...e.
00402150: C9 01 06 00 CA 00 C9 01 06 00 5A 01 C9 01 06 00 E...E.E...Z.E...
00402160: 26 01 C9 01 06 00 3F 01 C9 01 06 00 79 00 C9 01 &.E...?.E...y.E.
00402170: 06 00 4E 00 FA 01 06 00 2C 00 FA 01 06 00 AD 00 ..N.u...,u...-.
00402180: C9 01 06 00 94 00 9B 01 06 00 4D 02 BD 01 06 00 E.....M.?...
00402190: 1A 00 BD 01 00 00 00 00 08 00 00 00 00 00 01 00 ..?.....
004021A0: 01 00 00 00 10 00 B5 01 DB 01 41 00 01 00 01 00 .....μ.U.A....
004021B0: 50 20 00 00 00 00 96 00 C4 01 28 00 01 00 64 20 P .....A.(...d
004021C0: 00 00 00 00 91 00 01 00 2E 00 02 00 72 20 00 00 .....r ..
004021D0: 00 00 86 18 E1 01 06 00 02 00 00 00 01 00 48 02 ....a.....H.
004021E0: 09 00 E1 01 01 00 11 00 E1 01 06 00 19 00 E1 01 ..a.....a.....a.
004021F0: 0A 00 29 00 E1 01 10 00 31 00 E1 01 10 00 39 00 ..).a...1.a...9.
00402200: E1 01 10 00 41 00 E1 01 10 00 49 00 E1 01 10 00 a...A.a...I.a...
00402210: 51 00 E1 01 10 00 59 00 E1 01 10 00 61 00 E1 01 Q.a...Y.a...a.a.
00402220: 15 00 69 00 E1 01 10 00 71 00 E1 01 10 00 79 00 ..i.a...q.a...y.
00402230: E1 01 10 00 89 00 22 00 1A 00 81 00 E1 01 06 00 a.....".....a...
00402240: 2E 00 0B 00 32 00 2E 00 13 00 3B 00 2E 00 1B 00 ....2.....;....
00402250: 5A 00 2E 00 23 00 63 00 2E 00 2B 00 6E 00 2E 00 Z...#.c...+n...
00402260: 33 00 6E 00 2E 00 3B 00 6E 00 2E 00 43 00 63 00 3.n...;n...C.c.
00402270: 2E 00 4B 00 74 00 2E 00 53 00 6E 00 2E 00 5B 00 ..K.t...S.n...[.
00402280: 6E 00 2E 00 63 00 8C 00 2E 00 6B 00 B6 00 2E 00 n...c.....k.¶...
00402290: 73 00 C3 00 04 80 00 00 01 00 00 00 00 00 00 00 s.A.....
004022A0: 00 00 00 00 00 00 DB 01 00 00 04 00 00 00 00 00 .....U.....
004022B0: 00 00 00 00 00 00 1F 00 11 00 00 00 00 00 00 00 .....
004022C0: 00 48 65 6C 6C 6F 32 00 3C 4D 6F 64 75 6C 65 3E .Hello2.<Module>
004022D0: 00 6D 73 63 6F 72 6C 69 62 00 43 6F 6E 73 6F 6C .mscorlib.Consol
004022E0: 65 00 57 72 69 74 65 4C 69 6E 65 00 47 75 69 64 e.WriteLine.Guid

```

004022F0: 41 74 74 72 69 62 75 74 65 00 44 65 62 75 67 67 Attribute.Debugg
 00402300: 61 62 6C 65 41 74 74 72 69 62 75 74 65 00 43 6F ableAttribute.Co
 00402310: 6D 56 69 73 69 62 6C 65 41 74 74 72 69 62 75 74 mVisibleAttribut
 00402320: 65 00 41 73 73 65 6D 62 6C 79 54 69 74 6C 65 41 e.AssemblyTitleA
 00402330: 74 74 72 69 62 75 74 65 00 41 73 73 65 6D 62 6C ttribute.Assembl
 00402340: 79 54 72 61 64 65 6D 61 72 6B 41 74 74 72 69 62 yTrademarkAttrib
 00402350: 75 74 65 00 54 61 72 67 65 74 46 72 61 6D 65 77 ute.TargetFramew
 00402360: 6F 72 6B 41 74 74 72 69 62 75 74 65 00 41 73 73 orkAttribute.Ass
 00402370: 65 6D 62 6C 79 46 69 6C 65 56 65 72 73 69 6F 6E emblyFileVersion
 00402380: 41 74 74 72 69 62 75 74 65 00 41 73 73 65 6D 62 Attribute.Assemb
 00402390: 6C 79 43 6F 6E 66 69 67 75 72 61 74 69 6F 6E 41 lyConfigurationA
 004023A0: 74 74 72 69 62 75 74 65 00 41 73 73 65 6D 62 6C ttribute.Assembl
 004023B0: 79 44 65 73 63 72 69 70 74 69 6F 6E 41 74 74 72 yDescriptionAttr
 004023C0: 69 62 75 74 65 00 43 6F 6D 70 69 6C 61 74 69 6F ibute.Compilatio
 004023D0: 6E 52 65 6C 61 78 61 74 69 6F 6E 73 41 74 74 72 nRelaxationsAttr
 004023E0: 69 62 75 74 65 00 41 73 73 65 6D 62 6C 79 50 72 ibute.AssemblyPr
 004023F0: 6F 64 75 63 74 41 74 74 72 69 62 75 74 65 00 41 oductAttribute.A
 00402400: 73 73 65 6D 62 6C 79 43 6F 70 79 72 69 67 68 74 ssemblyCopyright
 00402410: 41 74 74 72 69 62 75 74 65 00 41 73 73 65 6D 62 Attribute.Assemb
 00402420: 6C 79 43 6F 6D 70 61 6E 79 41 74 74 72 69 62 75 lyCompanyAttribu
 00402430: 74 65 00 52 75 6E 74 69 6D 65 43 6F 6D 70 61 74 te.RuntimeCompat
 00402440: 69 62 69 6C 69 74 79 41 74 74 72 69 62 75 74 65 ibilityAttribute
 00402450: 00 48 65 6C 6C 6F 2E 65 78 65 00 53 79 73 74 65 .Hello.exe.Syste
 00402460: 6D 2E 52 75 6E 74 69 6D 65 2E 56 65 72 73 69 6F m.Runtime.Versio
 00402470: 6E 69 6E 67 00 50 72 6F 67 72 61 6D 00 53 79 73 ning.Program.Sys
 00402480: 74 65 6D 00 4D 61 69 6E 00 53 79 73 74 65 6D 2E tem.Main.System.
 00402490: 52 65 66 6C 65 63 74 69 6F 6E 00 48 65 6C 6C 6F Reflection.Hello
 004024A0: 00 2E 63 74 6F 72 00 53 79 73 74 65 6D 2E 44 69 ..ctor.System.Di
 004024B0: 61 67 6E 6F 73 74 69 63 73 00 53 79 73 74 65 6D agnostics.System
 004024C0: 2E 52 75 6E 74 69 6D 65 2E 49 6E 74 65 72 6F 70 .Runtime.Interop
 004024D0: 53 65 72 76 69 63 65 73 00 53 79 73 74 65 6D 2E Services.System.
 004024E0: 52 75 6E 74 69 6D 65 2E 43 6F 6D 70 69 6C 65 72 Runtime.Compiler
 004024F0: 53 65 72 76 69 63 65 73 00 44 65 62 75 67 67 69 Services.Debuggi
 00402500: 6E 67 4D 6F 64 65 73 00 61 72 67 73 00 4F 62 6A ngModes.args.Obj
 00402510: 65 63 74 00 00 19 48 00 65 00 6C 00 6C 00 6F 00 ect...H.e.l.l.o.
 00402520: 20 00 57 00 6F 00 72 00 6C 00 64 00 21 00 00 1F .W.o.r.l.d.!...
 00402530: 48 00 65 00 6C 00 6C 00 6F 00 20 00 53 00 74 00 H.e.l.l.o. .S.t.
 00402540: 75 00 64 00 65 00 6E 00 74 00 73 00 21 00 00 00 u.d.e.n.t.s.!...
 00402550: C7 0A 86 FF 2C 4B 2F 44 88 FB 31 13 78 F3 60 80 C..y,K/D.u1.xo`.
 00402560: 00 04 20 01 01 08 03 20 00 01 05 20 01 01 11 11
 00402570: 04 20 01 01 0E 04 20 01 01 02 04 00 01 01 0E 08
 00402580: B7 7A 5C 56 19 34 E0 89 05 00 01 01 1D 0E 03 00 -z\V.4a.....
 00402590: 00 01 08 01 00 08 00 00 00 00 00 1E 01 00 01 00
 004025A0: 54 02 16 57 72 61 70 4E 6F 6E 45 78 63 65 70 74 T..WrapNonExcept
 004025B0: 69 6F 6E 54 68 72 6F 77 73 01 08 01 00 07 01 00 ionThrows.....

```

004025C0: 00 00 00 0A 01 00 05 48 65 6C 6C 6F 00 00 05 01 .....Hello....
004025D0: 00 00 00 00 17 01 00 12 43 6F 70 79 72 69 67 68 .....Copyright
004025E0: 74 20 C2 A9 20 20 32 30 32 32 00 00 29 01 00 24 t A© 2022..).$.
004025F0: 31 63 66 65 63 64 66 62 2D 34 66 63 34 2D 34 35 1cfecdfb-4fc4-45
00402600: 64 30 2D 61 39 62 38 2D 31 31 33 37 63 35 62 37 d0-a9b8-1137c5b7
00402610: 38 39 63 63 00 00 0C 01 00 07 31 2E 30 2E 30 2E 89cc.....1.0.0.
00402620: 30 00 00 4D 01 00 1C 2E 4E 45 54 46 72 61 6D 65 0..M....NETFrame
00402630: 77 6F 72 6B 2C 56 65 72 73 69 6F 6E 3D 76 34 2E work,Version=v4.
00402640: 37 2E 32 01 00 54 0E 14 46 72 61 6D 65 77 6F 72 7.2..T..Framework
00402650: 6B 44 69 73 70 6C 61 79 4E 61 6D 65 14 2E 4E 45 kDisplayName..NE
00402660: 54 20 46 72 61 6D 65 77 6F 72 6B 20 34 2E 37 2E T Framework 4.7.
00402670: 32 00 00 00 00 00 00 00 B3 A3 7A C6 00 00 00 00 2.....??z?....
00402680: 02 00 00 00 91 00 00 00 AC 26 00 00 AC 08 00 00 .....¬&..¬...
00402690: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 10 00 00 00 .....
004026A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 52 53 44 53 .....RSDS
004026B0: 14 7C EA AE 08 C2 57 4E A7 33 FB A7 FA A0 20 F0 .|e®.AWN§3u$u. ?
004026C0: 01 00 00 00 44 3A 5C 41 64 65 6C 5C D0 9A D0 B0 ....D:\Adel\?.?°
004026D0: D1 84 D0 B5 D0 B4 D1 80 D0 B0 5C D0 9A D0 9F D0 N.?μ??N.?°\?.?.?
004026E0: 9E 5C D0 9B D0 B5 D0 BA D1 86 D0 B8 D0 B8 5C D0 .\?.?μ??N.????\?
004026F0: 9E D1 81 D0 B5 D0 BD D0 BD D0 B8 D0 B9 20 D1 81 .N.?μ???????? N.
00402700: D0 B5 D0 BC D0 B5 D1 81 D1 82 D1 80 5C D0 9B D0 ?μ??μN.N.N.\?.?
00402710: B5 D0 BA D1 86 D0 B8 D1 8F 20 32 37 5C 48 65 6C μ??N.??N. 27\Hel
00402720: 6C 6F 5C 48 65 6C 6C 6F 5C 6F 62 6A 5C 44 65 62 lo\Hello\obj\Deb
00402730: 75 67 5C 48 65 6C 6C 6F 2E 70 64 62 00 65 27 00 ug\Hello.pdb.e'.
00402740: 00 00 00 00 00 00 00 00 00 00 7F 27 00 00 00 20 00 .....!... .
00402750: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00402760: 00 00 00 00 00 71 27 00 00 00 00 00 00 00 00 00 .....q'.....
00402770: 00 00 00 5F 43 6F 72 45 78 65 4D 61 69 6E 00 6D ..._CorExeMain.m
00402780: 73 63 6F 72 65 65 2E 64 6C 6C 00 00 00 00 00 00 scoree.dll.....
00402790: 00 00 FF 25 00 20 40 00 ..y%. @.

```

Debug Directories

Time	Type	Size	RVA	Pointer
C67AA3B3	cv	91	000026AC	8AC Format: RSDS, {AEEA7C14-C208-4E57-A733-FBA7FAA020F0}, 1, D:\Adel\Кафедра\КПО\Лекции\Осенний семестр\Лекция 27\Hello\Hello\obj\Debug\Hello.pdb
00000000	repro	0	00000000	0

clr Header:

```

48 cb
2.05 runtime version
207C [ 5F8] RVA [size] of MetaData Directory

```


20003 flags

IL Only

32-Bit Required

32-Bit Preferred

6000001 entry point token

0 [0] RVA [size] of Resources Directory

0 [0] RVA [size] of StrongNameSignature Directory

0 [0] RVA [size] of CodeManagerTable Directory

0 [0] RVA [size] of VTableFixups Directory

0 [0] RVA [size] of ExportAddressTableJumps Directory

0 [0] RVA [size] of ManagedNativeHeader Directory

Section contains the following imports:

mscorlib.dll

402000 Import Address Table

402765 Import Name Table

0 time date stamp

0 Index of first forwarder reference

0 _CorExeMain

SECTION HEADER #2

.rsrc name

58C virtual size

4000 virtual address (00404000 to 0040458B)

600 size of raw data

A00 file pointer to raw data (00000A00 to 00000FFF)

0 file pointer to relocation table

0 file pointer to line numbers

0 number of relocations

0 number of line numbers

40000040 flags

Initialized Data

Read Only

RAW DATA #2

00404000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 02 00

00404010: 10 00 00 00 20 00 00 80 18 00 00 00 50 00 00 80P...

00404020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00

00404030: 01 00 00 00 38 00 00 80 00 00 00 00 00 00 00 008.....

00404040: 00 00 00 00 00 00 01 00 00 00 00 00 80 00 00 00

00404050: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00

00404060: 01 00 00 00 68 00 00 80 00 00 00 00 00 00 00 00h.....

00404070: 00 00 00 00 00 00 01 00 00 00 00 00 8C 03 00 00
 00404080: 90 40 00 00 FC 02 00 00 00 00 00 00 00 00 00 00 .@..u.....
 00404090: FC 02 34 00 00 00 56 00 53 00 5F 00 56 00 45 00 u.4...V.S._.V.E.
 004040A0: 52 00 53 00 49 00 4F 00 4E 00 5F 00 49 00 4E 00 R.S.I.O.N._.I.N.
 004040B0: 46 00 4F 00 00 00 00 00 BD 04 EF FE 00 00 01 00 F.O.....?i?....
 004040C0: 00 00 01 00 00 00 00 00 00 00 01 00 00 00 00 00
 004040D0: 3F 00 00 00 00 00 00 00 04 00 00 00 01 00 00 00 ?.....
 004040E0: 00 00 00 00 00 00 00 00 00 00 00 00 44 00 00 00D...
 004040F0: 01 00 56 00 61 00 72 00 46 00 69 00 6C 00 65 00 ..V.ar.File.
 00404100: 49 00 6E 00 66 00 6F 00 00 00 00 00 24 00 04 00 In.f.o.....\$...
 00404110: 00 00 54 00 72 00 61 00 6E 00 73 00 6C 00 61 00 ..T.r.a.n.s.l.a.
 00404120: 74 00 69 00 6F 00 6E 00 00 00 00 00 00 00 B0 04 t.i.o.n.....°.
 00404130: 5C 02 00 00 01 00 53 00 74 00 72 00 69 00 6E 00 \.....S.tr.in.
 00404140: 67 00 46 00 69 00 6C 00 65 00 49 00 6E 00 66 00 g.File.Inf.
 00404150: 6F 00 00 00 38 02 00 00 01 00 30 00 30 00 30 00 o...8....0.0.0.
 00404160: 30 00 30 00 34 00 62 00 30 00 00 00 1A 00 01 00 0.0.4.b.0.....
 00404170: 01 00 43 00 6F 00 6D 00 6D 00 65 00 6E 00 74 00 ..C.o.m.m.e.n.t.
 00404180: 73 00 00 00 00 00 00 00 22 00 01 00 01 00 43 00 s.....".....C.
 00404190: 6F 00 6D 00 70 00 61 00 6E 00 79 00 4E 00 61 00 o.m.p.a.n.y.N.a.
 004041A0: 6D 00 65 00 00 00 00 00 00 00 00 00 34 00 06 00 m.e.....4...
 004041B0: 01 00 46 00 69 00 6C 00 65 00 44 00 65 00 73 00 ..File.D.es.
 004041C0: 63 00 72 00 69 00 70 00 74 00 69 00 6F 00 6E 00 c.r.i.p.t.i.o.n.
 004041D0: 00 00 00 00 48 00 65 00 6C 00 6C 00 6F 00 00 00H.e.l.l.o...
 004041E0: 30 00 08 00 01 00 46 00 69 00 6C 00 65 00 56 00 0....File.V.
 004041F0: 65 00 72 00 73 00 69 00 6F 00 6E 00 00 00 00 00 e.r.s.i.o.n.....
 00404200: 31 00 2E 00 30 00 2E 00 30 00 2E 00 30 00 00 00 1...0...0...0...
 00404210: 34 00 0A 00 01 00 49 00 6E 00 74 00 65 00 72 00 4....I.n.t.e.r.
 00404220: 6E 00 61 00 6C 00 4E 00 61 00 6D 00 65 00 00 00 n.a.l.N.a.m.e...
 00404230: 48 00 65 00 6C 00 6C 00 6F 00 2E 00 65 00 78 00 H.e.l.l.o...e.x.
 00404240: 65 00 00 00 48 00 12 00 01 00 4C 00 65 00 67 00 e...H....L.e.g.
 00404250: 61 00 6C 00 43 00 6F 00 70 00 79 00 72 00 69 00 a.l.C.o.p.y.r.i.
 00404260: 67 00 68 00 74 00 00 00 43 00 6F 00 70 00 79 00 g.h.t...C.o.p.y.
 00404270: 72 00 69 00 67 00 68 00 74 00 20 00 A9 00 20 00 r.i.g.h.t..©..
 00404280: 20 00 32 00 30 00 32 00 32 00 00 00 2A 00 01 00 .2.0.2.2...*...
 00404290: 01 00 4C 00 65 00 67 00 61 00 6C 00 54 00 72 00 ..L.e.g.a.l.Tr.
 004042A0: 61 00 64 00 65 00 6D 00 61 00 72 00 6B 00 73 00 a.d.e.m.a.r.k.s.
 004042B0: 00 00 00 00 00 00 00 00 3C 00 0A 00 01 00 4F 00<.....O.
 004042C0: 72 00 69 00 67 00 69 00 6E 00 61 00 6C 00 46 00 r.i.g.i.n.a.l.F.
 004042D0: 69 00 6C 00 65 00 6E 00 61 00 6D 00 65 00 00 00 i.l.e.n.a.m.e...
 004042E0: 48 00 65 00 6C 00 6C 00 6F 00 2E 00 65 00 78 00 H.e.l.l.o...e.x.
 004042F0: 65 00 00 00 2C 00 06 00 01 00 50 00 72 00 6F 00 e...,.....P.r.o.
 00404300: 64 00 75 00 63 00 74 00 4E 00 61 00 6D 00 65 00 d.u.c.t.N.a.m.e.
 00404310: 00 00 00 00 48 00 65 00 6C 00 6C 00 6F 00 00 00H.e.l.l.o...
 00404320: 34 00 08 00 01 00 50 00 72 00 6F 00 64 00 75 00 4....P.r.o.d.u.
 00404330: 63 00 74 00 56 00 65 00 72 00 73 00 69 00 6F 00 c.t.V.e.r.s.i.o.

```

00404340: 6E 00 00 00 31 00 2E 00 30 00 2E 00 30 00 2E 00  n...1...0...0...
00404350: 30 00 00 00 38 00 08 00 01 00 41 00 73 00 73 00  0...8.....A.s.s.
00404360: 65 00 6D 00 62 00 6C 00 79 00 20 00 56 00 65 00  e.m.b.l.y. .V.e.
00404370: 72 00 73 00 69 00 6F 00 6E 00 00 00 31 00 2E 00  r.s.i.o.n...1...
00404380: 30 00 2E 00 30 00 2E 00 30 00 00 00 9C 43 00 00  0...0...0....C..
00404390: EA 01 00 00 00 00 00 00 00 00 00 00 00 EF BB BF 3C  e.....i»?<
004043A0: 3F 78 6D 6C 20 76 65 72 73 69 6F 6E 3D 22 31 2E  ?xml version="1.
004043B0: 30 22 20 65 6E 63 6F 64 69 6E 67 3D 22 55 54 46  0" encoding="UTF
004043C0: 2D 38 22 20 73 74 61 6E 64 61 6C 6F 6E 65 3D 22  -8" standalone="
004043D0: 79 65 73 22 3F 3E 0D 0A 0D 0A 3C 61 73 73 65 6D  yes"?>....<assem
004043E0: 62 6C 79 20 78 6D 6C 6E 73 3D 22 75 72 6E 3A 73  bly xmlns="urn:s
004043F0: 63 68 65 6D 61 73 2D 6D 69 63 72 6F 73 6F 66 74  chemas-microsoft
00404400: 2D 63 6F 6D 3A 61 73 6D 2E 76 31 22 20 6D 61 6E  -com:asm.v1" man
00404410: 69 66 65 73 74 56 65 72 73 69 6F 6E 3D 22 31 2E  ifestVersion="1.
00404420: 30 22 3E 0D 0A 20 20 3C 61 73 73 65 6D 62 6C 79  0">.. <assembly
00404430: 49 64 65 6E 74 69 74 79 20 76 65 72 73 69 6F 6E  Identity version
00404440: 3D 22 31 2E 30 2E 30 2E 30 22 20 6E 61 6D 65 3D  ="1.0.0.0" name=
00404450: 22 4D 79 41 70 70 6C 69 63 61 74 69 6F 6E 2E 61  "MyApplication.a
00404460: 70 70 22 2F 3E 0D 0A 20 20 3C 74 72 75 73 74 49  pp"/>.. <trustI
00404470: 6E 66 6F 20 78 6D 6C 6E 73 3D 22 75 72 6E 3A 73  nfo xmlns="urn:s
00404480: 63 68 65 6D 61 73 2D 6D 69 63 72 6F 73 6F 66 74  chemas-microsoft
00404490: 2D 63 6F 6D 3A 61 73 6D 2E 76 32 22 3E 0D 0A 20  -com:asm.v2">..
004044A0: 20 20 20 3C 73 65 63 75 72 69 74 79 3E 0D 0A 20  <security>..
004044B0: 20 20 20 20 20 3C 72 65 71 75 65 73 74 65 64 50  <requestedP
004044C0: 72 69 76 69 6C 65 67 65 73 20 78 6D 6C 6E 73 3D  rivileges xmlns=
004044D0: 22 75 72 6E 3A 73 63 68 65 6D 61 73 2D 6D 69 63  "urn:schemas-mic
004044E0: 72 6F 73 6F 66 74 2D 63 6F 6D 3A 61 73 6D 2E 76  rosoft-com:asm.v
004044F0: 33 22 3E 0D 0A 20 20 20 20 20 20 20 20 20 3C 72 65  3">.. <re
00404500: 71 75 65 73 74 65 64 45 78 65 63 75 74 69 6F 6E  requestedExecution
00404510: 4C 65 76 65 6C 20 6C 65 76 65 6C 3D 22 61 73 49  Level level="asI
00404520: 6E 76 6F 6B 65 72 22 20 75 69 41 63 63 65 73 73  nvoker" uiAccess
00404530: 3D 22 66 61 6C 73 65 22 2F 3E 0D 0A 20 20 20 20  ="false"/>..
00404540: 20 20 3C 2F 72 65 71 75 65 73 74 65 64 50 72 69  </requestedPri
00404550: 76 69 6C 65 67 65 73 3E 0D 0A 20 20 20 20 3C 2F  vileges>.. </
00404560: 73 65 63 75 72 69 74 79 3E 0D 0A 20 20 3C 2F 74  security>.. </t
00404570: 72 75 73 74 49 6E 66 6F 3E 0D 0A 3C 2F 61 73 73  rustInfo>..</ass
00404580: 65 6D 62 6C 79 3E 00 00 00 00 00 00 00 00  embly>.....

```

SECTION HEADER #3

.reloc name

C virtual size

6000 virtual address (00406000 to 0040600B)

200 size of raw data

1000 file pointer to raw data (00001000 to 000011FF)

0 file pointer to relocation table

0 file pointer to line numbers
0 number of relocations
0 number of line numbers
42000040 flags
 Initialized Data
 Discardable
 Read Only

RAW DATA #3

00406000: 00 20 00 00 0C 00 00 00 94 37 00 00 7..

BASE RELOCATIONS #3

2000 RVA, C SizeOfBlock
794 HIGHLOW 00402000
0 ABS

Summary

2000 .reloc
2000 .rsrc
2000 .text

2. Частичная компиляция. «Шитый код»

После внесения изменений компилируются только те части программы, которые были модифицированы после предыдущей компиляции.



Логически можно выделить два подхода к реализации языков программирования – трансляцию и интерпретацию. Между этими полюсами располагается целый спектр промежуточных подходов, состоящих в предварительном преобразовании входного текста программы в некоторый промежуточный язык с последующей интерпретацией получившегося кода. Чем ближе промежуточный язык к машинному языку, тем ближе данный подход к классической трансляции, а чем ближе он к исходному языку программирования, тем ближе этот подход к интерпретации.

Техника реализации схемы.

Техника «Шитый код» использовалась при реализации языка Форт (Чарльз Х. Мур, конец 60-х, начало 70-х гг.).

Шитый код особенно удобен для реализации виртуальных машин со стековой архитектурой: П-кода, Лиспа и, конечно, Форта.

Шитый код (threaded code) – один из способов реализации виртуальной машины при интерпретации языков программирования (наряду с байт-кодом).

Основное представление программы при использовании шитого кода – массив вызовов подпрограмм. Реализация шитого кода, способ хранения этих вызовов может быть различной. Этот код может обрабатываться интерпретатором или может быть простой последовательностью машинных инструкций вызова подпрограмм/процедур.

Известны четыре разновидности шитого кода: подпрограммная, прямая, косвенная и свернутая.

Во всех разновидностях шитого кода его интерпретатор должен обеспечивать выполнение трех действий, которые традиционно обозначаются так:

NEXT (следующий) – переход к интерпретации следующей ссылки в данной последовательности ссылок;

CALL (вызов) – переход в подпрограмму верхнего уровня, представленную в шитом коде;

RETURN (возврат) – возврат из подпрограммы верхнего уровня на продолжение интерпретации.

Подпрограммный код

Эта разновидность шитого кода по сути ничем не отличается от машинного кода. Это последовательность вызовов уже скомпилированных подпрограмм.

Программа имеет следующий вид:

```
call sub1;  
call sub2;  
call sub3;
```

Прямой шитый код

Этот код получается из подпрограммного, если из кода убрать вызовы call. В теле кода остаются только адреса подпрограмм (уступает подпрограммному по скорости исполнения, но дает выигрыш по объему памяти, необходимой для его размещения).

Вызов подпрограмм осуществляется с помощью простейшего адресного интерпретатора, занимающего несколько машинных команд (в некоторых процессорных архитектурах — одну).

```
call Interpretator;  
AddrSub1;  
AddrSub2;  
...  
AddrEXIT;  
...  
...  
Interpretator: машинный код, NEXT  
...  
Sub1: машинный код, NEXT  
...  
EXIT: машинный код, NEXT
```

Требуется разработка специального интерпретатора последовательности ссылок. Подпрограммы верхнего уровня должны начинаться машинными командами, выполняющими действие CALL (положить текущее значение указателя на стек возвратов, перевести указатель на начало последовательности

адресов данной подпрограммы, исполнить NEXT) и заканчиваться адресом подпрограммы RETURN (снять значение со стека возвратов и заслать его в указатель, исполнить NEXT).

В прямом шитом коде любое определение (например Sub1) начинается машинным кодом. Интерпретатор должен запомнить в стеке возвратов прошлое значение счётчика инструкций, который перемещается по коду, а текущим сделать свой адрес возврата.

NEXT — это последовательность, используемая вместо return. Если мы завершаем Sub1, то NEXT обращается к счётчику инструкций, изменяет его на размер кода и на следующем шаге уже исполняется первая машинная команда из Sub2.

EXIT — восстанавливает предыдущее значение счетчика команд и переходит по соответствующему адресу.

Косвенный шитый код

Отличается от прямого шитого кода тем, что тело кода начинается не вызовом интерпретатора, а *адресом*, где интерпретатор находится.

AddrInterpreter;

AddrSub1;

AddrSub2;

...

AddrEXIT;

...

...

Interpreter: Адрес машинного кода, машинный код, NEXT

...

Sub...: Адрес машинного кода, машинный код, NEXT

...

EXIT: Адрес машинного кода, машинный код, NEXT

Свёрнутый шитый код

Может, например, использоваться для сокращения размера кода, когда он имеет критическое значение. Он может быть как прямым, так и косвенным. Вместо прямых адресов подпрограмм и кодов в нем используются их свертки, которые, вообще говоря, короче этих адресов. Используя 2-байтные коды, можно использовать адресное пространство, значительно превышающее 64 килобайт.

Так, если мы знаем, что код и данные выровнены относительно размеров некоторого сегмента (например 16 байт), мы можем использовать в качестве свёрнутого адреса физический адрес, делённый на 16.

В ряде случаев для свёртки может использоваться адресная таблица. Шитый код представляет собой положение адреса в таблице. Интерпретатор читает из таблицы этот код и совершает переход по соответствующему адресу.

Байт-код можно рассматривать как специальный случай свёрнутого шитого кода с адресной таблицей.