Лабораторная работа №2. Программирование на языке MSIL

Цель: ознакомиться с синтаксисом языка IL, научиться применять базовые конструкции языка и отлаживать написанные приложения

Инструменты: дизассемблер ILDASM, ассемблер ILASM, отладчик Visual Studio

Задание 1. Создание простого процедурного приложения

1. Создайте в текстовом редакторе новый файл ***Lab2.il*** и введите минимальный фрагмент IL-кода, который может быть скомпилирован:

```
.assembly Lab2Assemly //Определение текущей сборки
{
    .ver 1:0:0:0 //Номер версии сборки
}
.method static void Main() //Главный метод программы
{
    .entrypoint //Текущий метод есть точкой входа
    ret //Возврат из метода
}
```

2. Откройте в Visual Studio окно «Командная строка разработчика» и убедитесь, что файл ***Lab2.il*** компилируется в новую сборку ***.exe**, введя в командной строке разработчика следующую команду:

>ilasm /exe *Lab2.il*

3. Проверьте с помощью утилиты командной строки **peverify.exe**, является ли скомпилированный файл правильно оформленным:

> peverify *lab2.exe*

4. Если скомпилированный файл не прошел проверку, вернитесь к исходному il-файлу для устранения ошибок и повторите п.п.2-3.
5. Запустите файл ***lab2.exe*** на выполнение и убедитесь в том, что программа запускается, хотя и не выполняет пока никаких действий.
6. Продумайте, какие функции необходимы для решения следующей задачи: запросить у пользователя два целых числа, найти их сумму и вывести результат на экран. Для решения этой задачи предлагается подготовить три вспомогательные функции:

Сигнатура функции	Назначение
int32 GetNumber(string varName)	Для организации диалога, обеспечивающего ввод операндов. Параметр varName – текстовая строка, после вывода которой ожидается ввод числа
int32 Sum(int32 x, int32 y)	Для вычисления суммы двух чисел

void SumPrint(int32 x, int32 y, int32 z) Для вывода результата вычислений

7. Дополните файл ***Lab2.il*** методом **GetNumber**():

```
.method static int32 GetNumber(string varName)
{
    //Загружаем в стек текст приглашения на ввод числа
    ldarg varName
    //Выводим приглашение на экран
    call void [mscorlib]System.Console::Write(string)
    //Ожидаем ввода числа
    call string [mscorlib]System.Console::ReadLine()
    //Преобразуем (если возможно) введенное значение в число
  call int32 [mscorlib]System.Int32::Parse(string)
    //Возвращаем полученное  число
    ret
}
```

8. Дополните файл ***Lab2.il*** методом **Sum** ():

```
.method static int32 Sum(int32 x, int32 y)
{
    ldarg x //Загружаем в стек аргумент x
    ldarg y //Загружаем в стек аргумент y
    add   // Находим сумму и помещаем результат в стек
    ret //Возвращаем результат
}
```

9. Дополните файл ***Lab2.il*** методом **SumPrint** ():

```
.method static void SumPrint(int32 x, int32 y, int32 z)
{
    ldstr "{0} + {1} = {2}" //Задаем строку формата для вывода данных

  ldarg x //Загружаем в стек аргумент x
  box int32 //Упаковываем x (нужно для WriteLine)
```

```
    ldarg y //Загружаем в стек аргумент y
    box int32 //Упаковываем y

    ldarg z //Загружаем в стек аргумент z
    box int32 //Упаковываем z

      //Выводим отформатированную строку
    call void [mscorlib]System.Console::WriteLine(string, object, object, object)

    ret //Выход из функции
}
```

10. Модифицируйте метод **Main** () следующим образом:

```
.method static void Main()
{
      //Объявляем локальные переменные
    .locals init (int32 x, //операнд 1
                  int32 y, //операнд 2
                  int32 z) //результат операции

      ldstr "x="  //Загружаем в стек текст приглашения на ввод операнда 1
      call int32 GetNumber(string) //Получаем значение операнда 1
      stloc x //Сохраняем значение операнда 1 в локальной переменной x

      ldstr "y="  //Загружаем в стек текст приглашения на ввод операнда 2
      call int32 GetNumber(string) //Получаем значение операнда 2
      stloc y //Сохраняем значение операнда 2 в локальной переменной y

      ldloc x //Загружаем в стек локальную переменную x
      ldloc y //Загружаем в стек локальную переменную y
      call int32 Sum(int32 x, int32 y) //Вычисляем сумму x и y
      stloc z  //Сохраняем результат в локальной переменной z
```

```
    ldloc x //Загружаем в стек локальную переменную x

    ldloc y //Загружаем в стек локальную переменную y

    ldloc z //Загружаем в стек локальную переменную z

    call void SumPrint(int32 x, int32 y, int32 z) //Выводим результат


    .entrypoint //Текущий метод есть точкой входа

    ret //Возврат из метода

}
```

11. Рекомпелируйте файл ***Lab2.il*** в сборку ***.exe**, введя в командной строке разработчика следующую команду:

>ilasm /exe *Lab2.il*

12. Проверьте с помощью утилиты командной строки **peverify.exe**, является ли скомпилированный файл правильно оформленным:

> peverify *Lab2.exe*

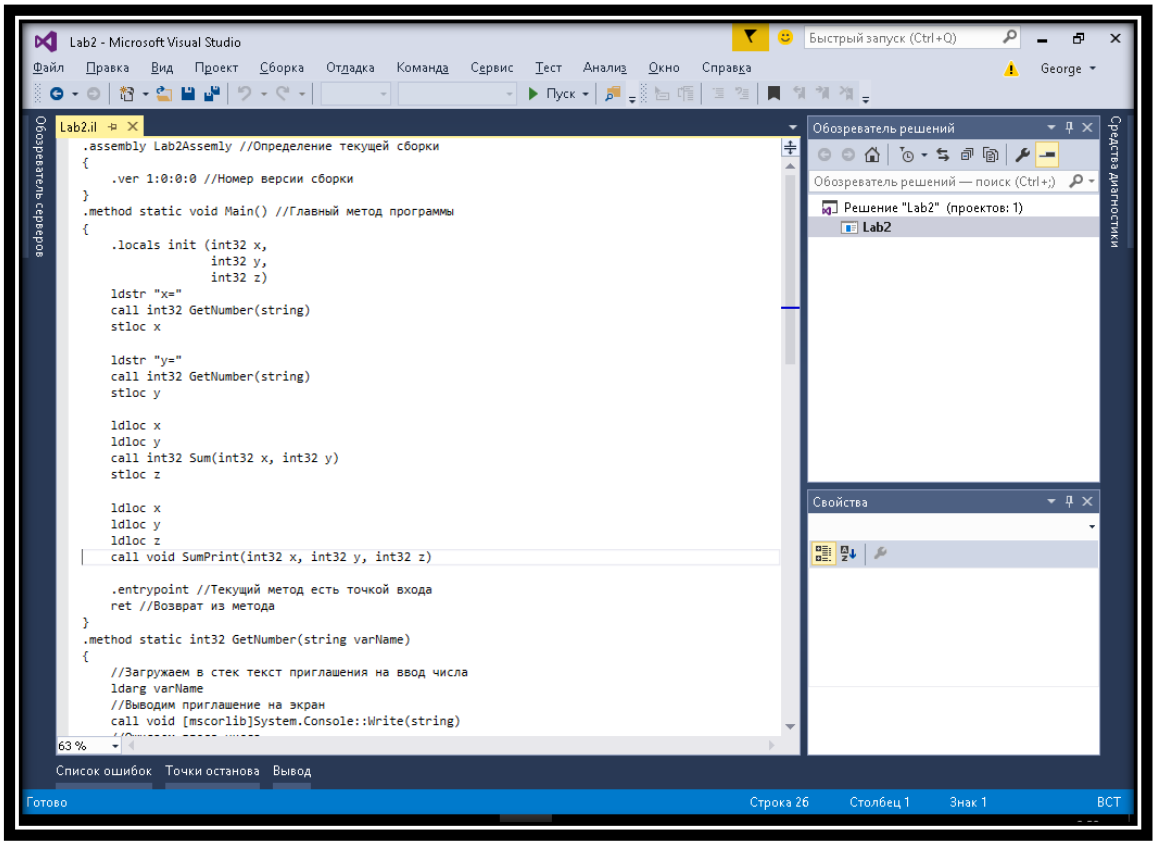
13. Если скомпилированный файл не прошел проверку, вернитесь к исходному il-файлу для устранения ошибок и повторите п.п.11-12. Для поиска ошибок также можно воспользоваться отладчиком Visual Studio, как это описано в Задании 2 этой лабораторной работы.
14. Запустите файл ***Lab2.exe*** на выполнение и убедитесь в работоспособности программы.

Задание 2. Отладка приложения в отладчике Visual Studio

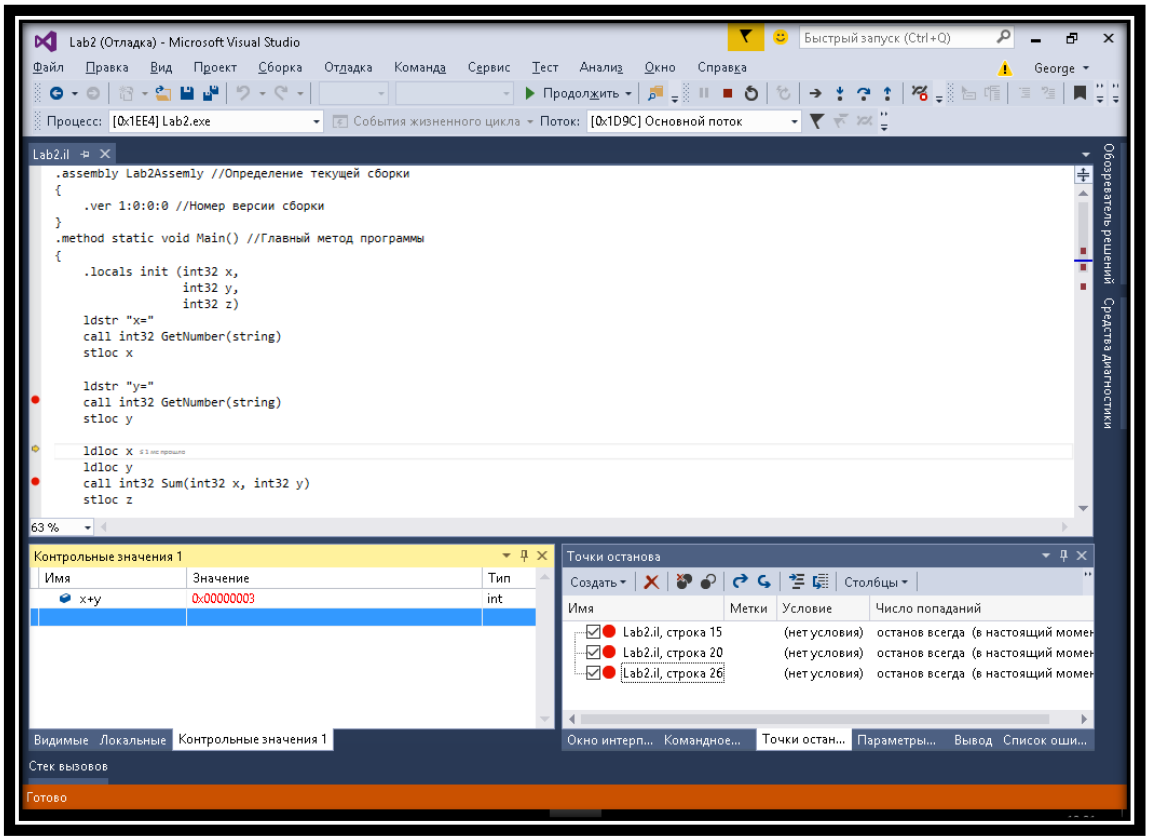
1. Создайте *файл базы данных программы* (файл с расширением .pdb), также называемый *файлом символов*. Такой PDB-*файл* (program database) сопоставляет идентификаторы, созданные в исходных файлах для классов, методов и другого кода, с идентификаторами, которые используются в скомпилированных исполняемых файлах проекта. PDB-файл также сопоставляет операторы в исходном коде с инструкциями выполнения в исполняемых файлах. Получить этот файл можно компилируя исходный код файла *.il, используя **ildasm.exe** с переключателем **debug**:

>ilasm *Lab2*.il /debug

2. Убедитесь, что файлы ***Lab2.pdb*** и ***Lab2.exe*** находятся в одном каталоге: по умолчанию отладчик ищет PDB-файл в месте расположения exe- (или dll-) файла.
3. Запустите Visual Studio.
4. С помощью команды **Файл-Открыть-Решение или проект…(Ctrl+Shift+O)** откройте файл ***Lab2.exe*** .
5. С помощью команды **Файл-Открыть-Файл …(Ctrl +O)** откройте файл ***Lab2.il*** .
6. Убедитесь, что рабочее окно Visual Studio приобрело следующий вид:



7. Сохраните текущее решение Visual Studio с помощью команды **Файл-Сохранить <*.sln> …(Ctrl +S).** Тогда в следующий раз для перехода к отладке достаточно открыть файл сохраненного решения (*.sln) вместо выполнения п.п.4-5.
8. Выполните пошаговую отладку приложения, используя традиционные средства отладки - точки останова, окна контрольных значений, интерпретации, командное окно и т.п.:



Задание 3. Создание объектно-ориентированного приложения

1. Откройте в текстовом редакторе файл ***Lab2.il*** и дополните его фрагментом кода, который создает класс по имени **Operation** вместе с конструктором класса:

```
.class public Operation
{
    .method void .ctor() //Конструктор класса
    {
            .maxstack 1
            ldarg.0
            call instance void [mscorlib]System.Object::.ctor()
            ret
    }
}
```

2. Переместите в созданный класс подготовленные ранее методы **GetNumber**(), **Sum**() и **SumPrint**():

```
.class Operation
{
    .method static int32 GetNumber(string varName) {//…}
    .method static int32 Sum(int32 x, int32 y) {//…}
    .method static void SumPrint(int32 x, int32 y, int32 z) {//…}
    .method void .ctor() {//…}
}
```

Здесь для краткости символами {//…} обозначено тело методов.

3. Модифицируйте метод Main() следующим образом:

```
.method static void Main()
{
  .locals init (int32 x,
            int32 y,
            int32 z,
            class Operation obj)//Обявление переменной типа Operation

    newobj void Operation::.ctor() //Создание экземпляра класса Operation
    stloc obj //Сохранение экземпляра класса Operation в переменной
```

```
    ldloc obj //Загрузка переменной типа Operation в стек

    ldstr "x="
    call int32 Operation::GetNumber(string)//Вызов метода класса Operation
    stloc x

    ldstr "y="
    call int32 Operation::GetNumber(string)//Вызов метода класса Operation
    stloc y

    ldloc x
    ldloc y
    call int32 Operation::Sum(int32 x, int32 y)//Вызов метода класса Operation
    stloc z

    ldloc x
    ldloc y
    ldloc z
    call void Operation::SumPrint(int32 x, int32 y, int32 z)//Вызов метода класса Operation

    pop //Удаление из стека переменной типа Operation

    .entrypoint
    ret
}
```

4. Рекомпелируйте файл ***Lab2.il*** в сборку ***.exe**, введя в командной строке разработчика следующую команду:

>ilasm /exe *Lab2.il*

5. Проверьте с помощью утилиты командной строки **peverify.exe**, является ли скомпилированный файл правильно оформленным:

>peverify *Lab2.exe*

6. Если скомпилированный файл не прошел проверку, вернитесь к исходному il-файлу для устранения ошибок и повторите п.п.4-5. Для поиска ошибок также можно воспользоваться отладчиком Visual Studio, как это описано в Задании 2 этой лабораторной работы.
7. Запустите файл ***Lab2.exe*** на выполнение и убедитесь в работоспособности программы.

Задание 4. Построение внешней сборки и ее использование

1. Создайте в текстовом редакторе новый файл ***OpLib.il*** (Operation Library) и объявите в нем имя внешней сборки:

```
.assembly OpLib {.ver 1:0:0:0 }
```

2. Переместите фрагмент кода с описанием класса Operation из файла ***Lab2.il*** в созданный файл.
3. Расширьте области видимости класса и методов с помощью атрибута **public**:

```
.assembly OpLib {.ver 2:0:0:0 }

.class public Operation

{

    .method public static int32 GetNumber(string varName) {//…}

    .method public static int32 Sum(int32 x, int32 y) {//…}

    .method public static void SumPrint(int32 x, int32 y, int32 z) {//…}

}
```

Здесь, как и ранее, для краткости символами {//…} обозначено тело методов.

4. Скомпелируйте файл ***OpLib.il*** в сборку ***.dll**, введя в командной строке разработчика следующую команду:

>ilasm /dll *Lab2.il*

5. Проверьте с помощью утилиты командной строки **peverify.exe**, является ли скомпилированный файл правильно оформленным:

>peverify *Lab2.exe*

6. Если скомпилированный файл не прошел проверку, вернитесь к исходному il-файлу для устранения ошибок и повторите п.п.4-5. Для поиска ошибок также можно воспользоваться отладчиком Visual Studio, как это описано в Задании 2 этой лабораторной работы.
7. Модифицируйте файл, добавив в него ссылку на внешнюю сборку и выполнив замену имени класса **Operation** на полное **[OpLib]Operation**:

```
.assembly extern OpLib {.ver 2:0:0:0 }//Ссылка на внешнюю сборку

.assembly Lab2Assemly {.ver 2:0:0:0 }

.method static void Main()

{

  .locals init (int32 x,

           int32 y,
```

```
            int32 z,
              class [OpLib]Operation obj)

    newobj instance void [OpLib]Operation::.ctor()
    stloc obj

    ldloc obj

    ldstr "x="
    call int32 [OpLib]Operation::GetNumber(string)
    stloc x

    ldstr "y="
    call int32 [OpLib]Operation::GetNumber(string)
    stloc y

    ldloc x
    ldloc y
    call int32 [OpLib]Operation::Sum(int32 x, int32 y)
    stloc z

    ldloc x
    ldloc y
    ldloc z
    call void [OpLib]Operation::SumPrint(int32 x, int32 y, int32 z)

    pop

    .entrypoint
    ret
}
```

8. Рекомпелируйте файл ***Lab2.il*** в сборку ***.exe**, введя в командной строке разработчика следующую команду:

>ilasm /exe *Lab2.il*

9. Проверьте с помощью утилиты командной строки **peverify.exe**, является ли скомпилированный файл правильно оформленным:

>peverify *Lab2.exe*

10. Если скомпилированный файл не прошел проверку, вернитесь к исходному il-файлу для устранения ошибок и повторите п.п.8-9. Для поиска ошибок также можно воспользоваться отладчиком Visual Studio, как это описано в Задании 2 этой лабораторной работы.
11. Запустите файл ***Lab2.exe*** на выполнение и убедитесь в работоспособности программы.

Задание 5. Построение внешней сборки и ее использование

1. Модифицируйте файл библиотеки ***OpLib.dll***, расширив ее методами для выполнения операций вычитания (sub), умножения (mul), деления (div) целых чисел.
2. Модифицируйте исполняемый файл ***Lab2.exe***, обеспечив вычисление левой и правой части одного из следующих тождеств:

Вариант	Тождество
1	$a^2-b^2 = (a-b)(a+b)$
2	$a^2-2ab+b^2 = (a-b)^2$
3	$a^2+2ab+b^2 = (a+b)^2$
4	$a^3+b^3 = (a+b)(a^2-ab+b^2)$
5	$a^3-b^3 = (a-b)(a^2+ab+b^2)$
6	$(a+b)^3=a^3+3a^2b+3ab^2+b^3$
7	$(a-b)^3=a^3-3a^2b+3ab^2-b^3$

Требование к отчету по лабораторной работе

Отчет должен содержать титульный лист с указанием темы и цели работы, результатов выполнения задания 5, а также выводов по работе.

Контрольные вопросы

1. Какой минимальный фрагмент IL-кода может быть скомпилирован?
2. Как в MSIL отмечается точка входа программы?
3. Как описываются функции в IL-коде?
4. Как описывается вызов метода класса в IL-коде?
5. Как в MSIL создать локальную переменную, сохранить в ней значение, загрузить в стек?
6. Как обеспечить возможность отладки IL-кода в Visual Studio?
7. Каково назначение PDB-файлов? Как они могут быть получены?
8. Как объявляется класс в IL-коде? Как описывается его конструктор?
9. Как объявляется метод класса в IL-коде? Как его вызвать?
10. Как создается внешняя сборка? Как она подключается к приложению?