P/Invoke

- P/Invoke = Platform Invoke
- Allows managed code to call unmanaged functions in COM objects, C/C++ DLLs, etc.
  - e.g. access to Win32 API
- To declare unmanaged functions
  - Use DllImport attribute and static extern
    ```
    [DllImport("kernel32.dll")]
    static extern int GetProcessHeap()
    ```
using System.Runtime.InteropServices;
namespace HelloWorld {
    class MyClass {
        [DllImport("user32.dll", CharSet=CharSet.Ansi)]
        static extern int MessageBox(int hwnd,
                                       string msg, string caption, int t);

        public static void Main() {
            MessageBox(0, "Hello World!", "Caption", 0);
        }
    }
}
Steps in P/Invoke

- Locates implementing DLL
- Loads DLL into memory
- Finds function address
- Pushes args on stack, marshalling data
- Transfers control to unmanaged code
P/Invoke Callbacks

- Unmanaged code can call back to managed code
  - Unmanaged parameter is function pointer
  - In managed code, must supply parameter as delegate
  - P/Invoke creates callback thunk
    - Passes address of thunk as callback parameters
public class SampleClass {
    delegate bool CallBack(int hwnd, int lParam);

    [DllImport("user32.dll")]
    static extern int EnumWindows(CallBack x,
        int lParam);

    // report the window handle
    public bool Report(int hwnd, int lParam) {
        Console.Write("Window handle is " + hwnd);
        return true;
    }

    public static void Main() {
        CallBack myCallBack = new CallBack(Report);
        EnumWindows(myCallBack, 0);
    }
}
C++/CLI

- Write managed C++ code
  - Compile with /clr
  - Generates CIL from C++
  - new keywords
    - __gc, __box, __typeof, __interface, __property
- Very useful for native access to C++ libraries
  - Build a “managed wrapper”
CIL

- Recall: two stage compilation
  - C# compiler: C# → CIL code
  - Just-in-time (JIT) compiler: CIL → native code

- Common Intermediate Language
  - Very close to C#
  - Define classes, structs, inheritance, methods
  - Assembly-like statements
CIL

• Stack language
  – Instead of registers, everything is from stack
  – Main operations take operands from stack
  – e.g.
    ```
    int a = 137;
    int b = 1;
    int k = a + b;
    ```
Hello World Example

.assembly extern mscorlib {} //automatically added
.assembly hello {}
.class Program {
    .method static public void Main() cil managed {
        .entrypoint //designates this method as the entry pt
        .locals init (string name) //create a local var
        ldstr "World" //load the string onto eval stack
        stloc.0 //store the string into the first local var
        ldstr "Hello, {0}!"
        ldloc name //load local var onto eval stack
        call void [mscorlib] System.Console:::WriteLine(
            string, object) //call method with stack items as params
        ret
    }
}
CIL Directives

- `.assembly`
- `.class`
  - Define any type
  - Extends: extend some other type
    - If extend `System.ValueType`, then value type
- `.method`, `.field`, `.property`, `.event`
- `.locals`: names and types for local vars
- `.entrypoint`
- `.maxstack`
Load/Store Operations

- **ldloc/stloc**
  - Pushes contents of local var (or index) onto stack
  - Pops and stores in local var (or index)
- **ldc**
  - ldc.i4 50000
  - ldc.i4 1
  - ldc.i4.ml
- **ldnull**
- **ldfld/stfld**
  - ldsfld int32 A::fielda
Load/Store Example

.locals init ([1] int32 a, [0] int32 b)
ldc.i4.5
stloc.0 ldc.i4 10
stloc.1
ldloc a
call void [mscorlib]
  System.Console:::WriteLine(int32)
ldloc b
call void [mscorlib]
  System.Console:::WriteLine(int32)
CIL Operations

- **Integer operations**
  - add, mul, sub, div, rem, neg

- **Boxing**
  - Removes the value type from stack
  - Creates an object on the managed heap that boxes the value type
  - Places a reference to the newly created object back on the evaluation stack

- **Data type conversions**: conv.*
CIL Operations

- **Construct objects**
  - newobj instance void A::.ctor()

- **Invoke functions**
  - call void [mscorlib]
    System.Console::WriteLine(string, object)
  - call instance int32 A::F()
  - callvirt instance int32 A::G()
Control Flow Operations

- **ceq/cgt/clt**
  - Pop top two elements of stack
  - Check =, >, <
  - Push true or false onto stack
- **br/beq/bne/bgt/blt/brfalse/brtrue**
  - Do the comparison and jump
  - Use to implement structured control flow
Control Flow Example

 ldc.i4.3  
 ldc.i4.1  
 cgt  
 brtrue greater  
 ldstr "{0} is less than or equal {1}"  
 br end  

greater:  
 ldstr "{0} is greater than {1}"  
 end:  
 ldc.i4.3  
 box int32  
 ldc.i4.1  
 box int32  
 call void [mscorlib]  
 System.Console:::WriteLine(string, object, object)
• Roundtripping
  - ildasm /out=new_program.il program.exe
  - Edit the CIL code in new_program.il
  - ilasm newadd.il