Guida completa

Assembly x86 (32 bit) + .NET solo da terminale (Windows)

Questa guida mostra **passo-passo** come: - lavorare in **assembler 32 bit (x86)** - creare una **DLL nativa** in ASM - usarla da **.NET (C#)** - **senza Visual Studio IDE**, **solo terminale**

Lo stile e l'approccio sono equivalenti a quelli della guida UVA (Evans), ma **interamente CLI**.

---

0. Requisiti e limiti

Requisiti

- **Windows** (obbligatorio)
- **.NET SDK** (es. net8.0)
- **MASM 32 bit** (`ml.exe`)
- **link.exe** (Microsoft linker)

  È sufficiente installare *Build Tools for Visual Studio*. Non serve l'IDE.

Limiti

- Solo **x86 / 32 bit**
- Solo **DLL native + P/Invoke**
- Debug ASM limitato

---

1. Aprire il terminale corretto

Apri **Developer Command Prompt for VS** (serve solo per PATH):

```
dotnet --version
ml
link
```

Se tutti rispondono, l'ambiente è pronto.

---

2. Creazione del progetto .NET (CLI)

```
mkdir AsmDotNet
cd AsmDotNet
dotnet new console
```

---

3. Forzare .NET a 32 bit (OBBLIGATORIO)

Apri il file di progetto:

```
notepad AsmDotNet.csproj
```

Sostituisci il contenuto:

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net8.0</TargetFramework>
    <PlatformTarget>x86</PlatformTarget>
  </PropertyGroup>
</Project>
```

> Senza `x86` otterrai **BadImageFormatException**.

---

4. Scrivere il codice Assembly 32 bit

Crea il file:

```
notepad AsmFunctions.asm
```

Contenuto (stdcall, stile universitario):

```
.386
.model flat, stdcall
option casemap:none

PUBLIC AddNumbers

.code
AddNumbers PROC a:DWORD, b:DWORD
    mov eax, a
    add eax, b
    ret
AddNumbers ENDP
END
```

Note importanti

- `stdcall` ⇒ lo stack è gestito dalla funzione
- il valore di ritorno è in `EAX`

5. Compilazione dell’assembly (x86)

```
ml /c /coff AsmFunctions.asm
```

Risultato:

```
AsmFunctions.obj
```

---

6. Problema classico: DllMain mancante

Se linki senza accorgimenti otterrai:

```
LNK2001: __DllMainCRTStartup@12
```

Motivo

- in ASM puro **non esiste** DllMain
- il linker se lo aspetta

Soluzione corretta

Usare `/NOENTRY`.

---

7. Problema classico: nome esportato (stdcall)

Con `stdcall`, la funzione:

```
AddNumbers
```

viene esportata come:

```
_AddNumbers@8
```

.NET **non la trova** se cerca `AddNumbers`.

Soluzione corretta: file `.def`

---

8. Creare il file DEF

```
notepad AsmFunctions.def
```

Contenuto:

```
LIBRARY AsmFunctions
EXPORTS
    AddNumbers=_AddNumbers@8
```

Questo esporta un **nome pulito**.

---

9. Creazione della DLL x86

```
link /dll /noentry /machine:x86 ^
     /def:AsmFunctions.def ^
     /out:AsmFunctions.dll AsmFunctions.obj
```

Risultato:

```
AsmFunctions.dll
```

---

10. Verifica delle esportazioni (opzionale)

```
dumpbin /exports AsmFunctions.dll
```

Output atteso:

```
AddNumbers
```

---

11. Codice C# (.NET)

Apri:

```
notepad Program.cs
```

Contenuto:

```
using System;
using System.Runtime.InteropServices;

class Program
{
    [DllImport("AsmFunctions.dll", CallingConvention =
CallingConvention.StdCall)]
    public static extern int AddNumbers(int a, int b);

    static void Main()
    {
        Console.WriteLine(AddNumbers(5, 7));
    }
}
```

---

12. Build del progetto

```
dotnet build
```

---

13. Copiare la DLL nella cartella di output

```
copy AsmFunctions.dll bin\Debug\net8.0
```

---

14. Esecuzione

```
dotnet run
```

Output:

```
12
```

---

15. Struttura finale del progetto

```
AsmDotNet/
├── AsmFunctions.asm
├── AsmFunctions.obj
├── AsmFunctions.def
```

```
├── AsmFunctions.dll
├── Program.cs
├── AsmDotNet.csproj
└── bin/
```

---

16. Errori comuni e soluzioni rapide

Errore	Causa	Soluzione
BadImageFormatException	.NET non x86	`<PlatformTarget>x86</PlatformTarget>`
EntryPointNotFound	nome decorato	usare `.def`
LNK2001 DllMain	nessun entry	`/NOENTRY`
Crash	calling convention	stdcall coerente
DllNotFound	DLL non copiata	copiare in `bin`

---

17. Regole d'oro (x86)

- ASM e .NET **devono essere entrambi 32 bit**
- `stdcall` ⇒ nome decorato `_Func@N`
- sempre `.def` per esportazioni pulite
- stack alignment **fondamentale**

---

18. Estensioni possibili

- passare a **cdecl**
- esportare più funzioni
- passare array e stringhe
- automatizzare `ml` e `link` nel `.csproj`
- confronto MASM vs NASM

Conclusione

Con questa configurazione: - lavori **esattamente come nella guida UVA** - usi **assembler 32 bit reale** - controlli ogni fase - **nessuna dipendenza da Visual Studio IDE**

Questa è la configurazione didatticamente più corretta per capire davvero il funzionamento ASM ↔ .NET.