CYBERSECURITY LAB #5

Giacomo Gori – Tutor didattico

g.gori@unibo.it

Exercise

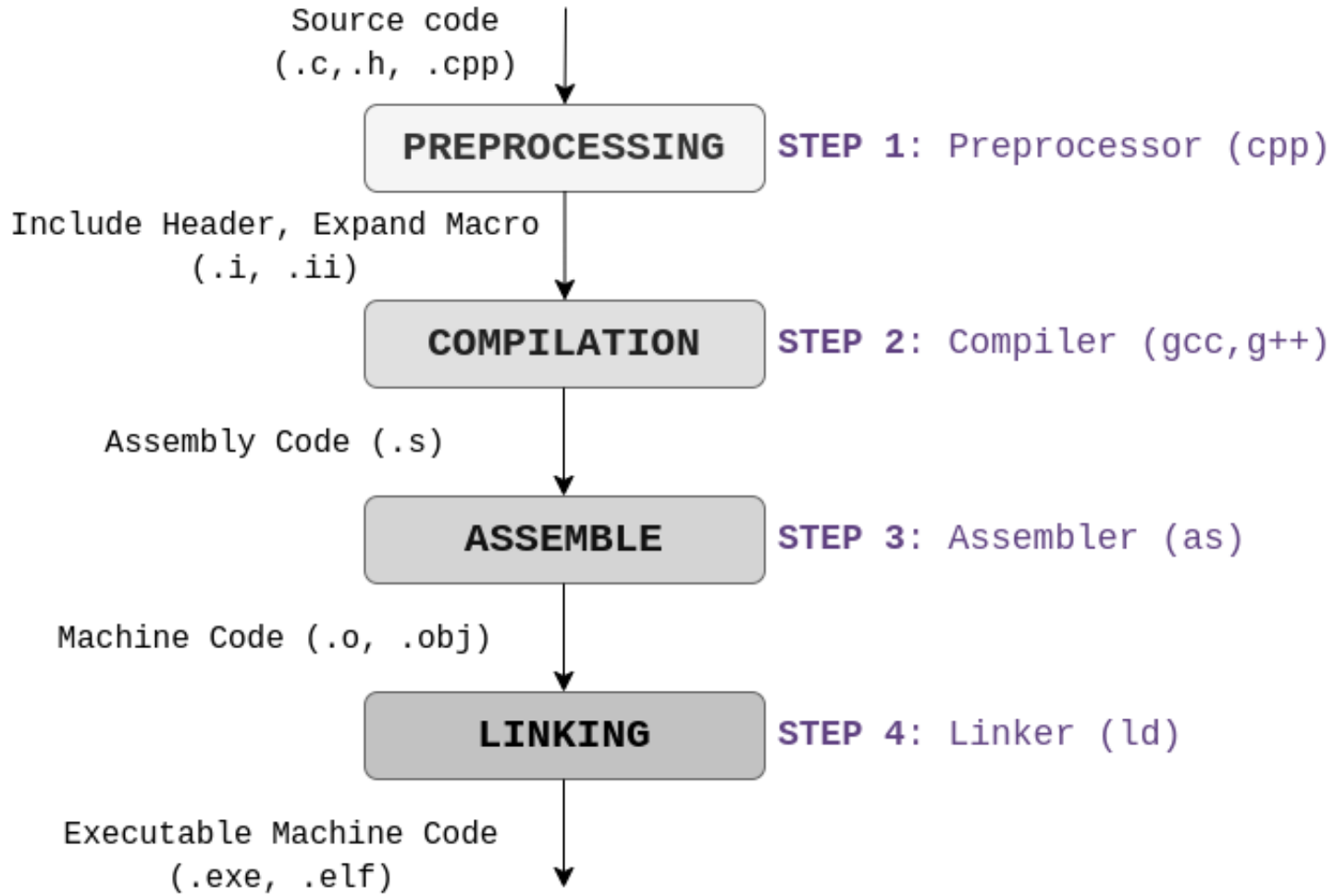
Patch the program to see the flag

Write a **small** report containing **the steps and the flag**

Remember: write name, surname and the number of the lab session on the report!

How are C programs compiled?

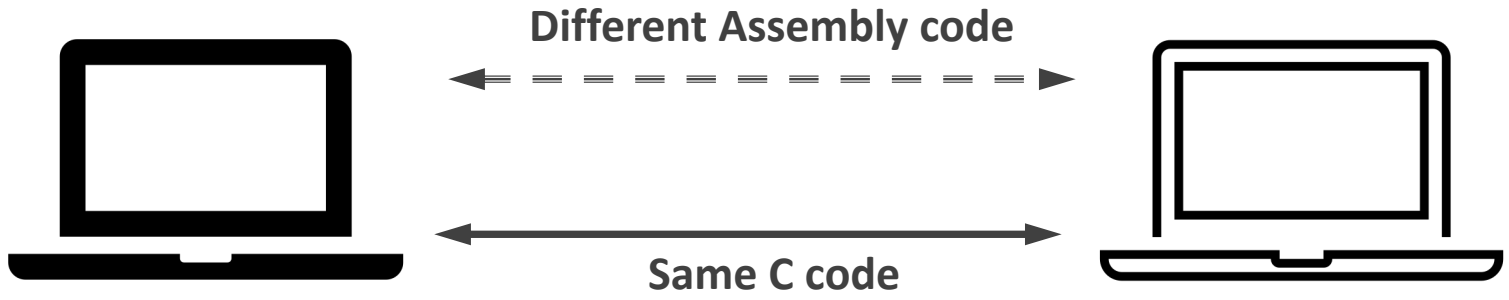
Steps of the C compiling



Let's dive down the various **compile steps** to better understand what this is all about

Compilation: assembly

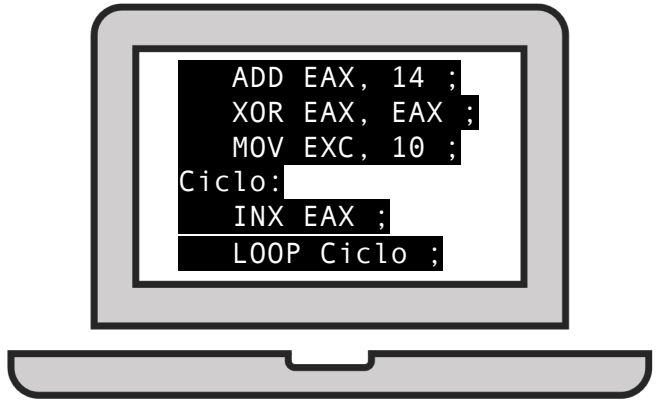
An intermediate step from the high level code (es: C) and the low level machine code.



Assembly code

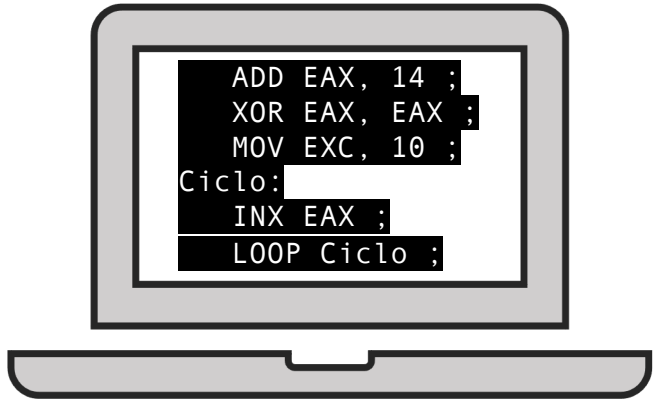
Translation of high level code into «*simple*» instruction on registers

- The ISA (Instruction Set Architecture) defines which instruction you can do
- Different CPU, different ISA :(
  - Es: RISC vs CISC, x86-32 and x86-64



Why Assembly?

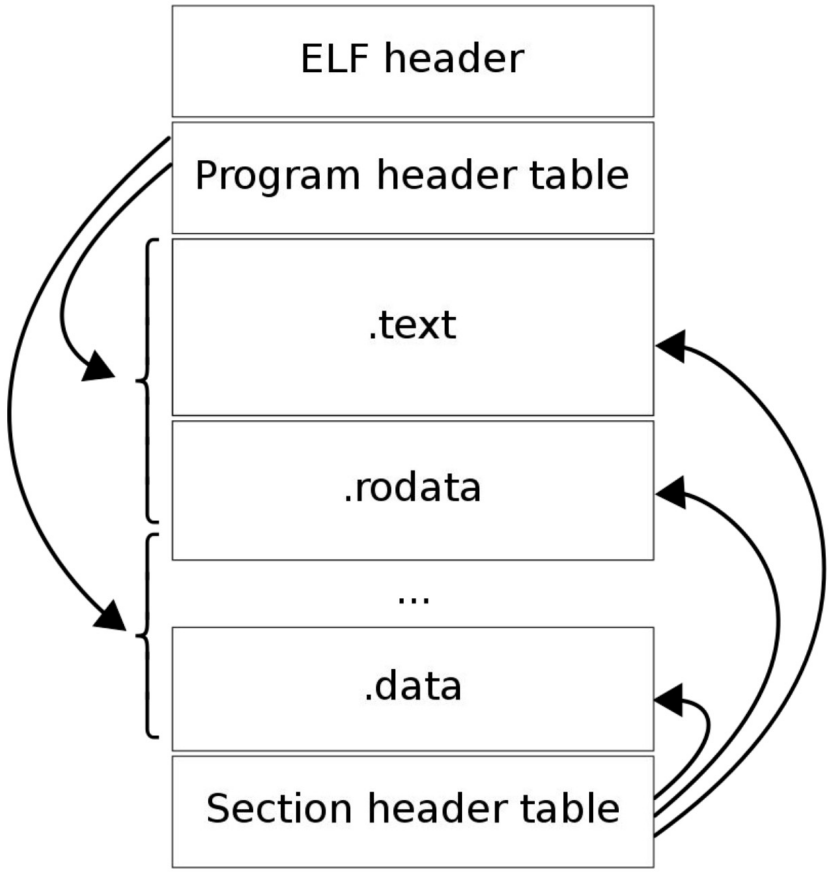
- High level languages are **complex** and would require extreme **complex and expensive CPU** architectures
- Instead: same high level code for different machines, then compilers create the specific assembly
  - **Portability** :)



Assemble + linker: machine code

In Linux, after the linking, machine code is serialized in a structured file which is formatted in the **Executable and Linkable Format** (ELF).

- Mainly divided in two parts:
  - Header
  - File data



ELF Header

`readelf -h ./(nome file)`

```
└─$ readelf -h a.out
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Position-Independent Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x1050
  Start of program headers:          64 (bytes into file)
  Start of section headers:          13968 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 30
```

Can we go “the other way”,
so to DEcompile?

Going back:
C decompiling

Taking a elf/exe file and bringing back the **source code** involves two main steps:

- 1° step: **disassembly** (easy)
- 2° step: **decompile** (hard)

Ghidra

It’s a free and open source **reverse engineering tool** by NSA.

We will use it to **disassembly and decompile binaries**, obtaining C code.

Installing Ghidra

Install jdk:

```
sudo apt update
sudo apt install default-jre
sudo apt install default-jdk
```

Download the latest release from
https://github.com/NationalSecurityAgency/ghidra/releases

Run Ghidra:

```
./ghidraRun
```

Using Ghidra

Let’s open Ghidra and try to **decompile a simple binary**.

To do that:

- create a new project
- import the binary file
- double click on it to view the disassembled code.
- open the functions to see them «decompiled».

Let’s see the **differences** between the original code and the decompiled one.

```c
#include <stdio.h>

int main(int argc, char * argv[]){
        int a = 5;
        printf("%d",a);
        printf("\n%s", argv[0]);
        printf("%d", argc);
        return 0;
}
```

```c
undefined8 main(uint param_1,undefined8 *param_2)

{
  printf("%d",5);
  printf("\n%s",*param_2);
  printf("%d",(ulong)param_1);
  return 0;
}
```

SPEEDS UP PRINTING

Download the executable file from Virtuale and **try to patch it** to make it print the flag.... **quicker**!