

An Introduction to Assembler Languages



Paolo PRINETTO
Politecnico di Torino (Italy)
Univ. of Illinois at Chicago, IL (USA)
CINI Cybersecurity Nat. Lab. (Italy)

Paolo.Prinetto@polito.it

www.consorzio-cini.it
www.comitato-girotondo.org

License Information

This work is licensed under the Creative Commons BY-NC License



To view a copy of the license, visit:
http://creativecommons.org/licenses/by-nc/3.0/legalcode

Rel. 17.05.2019
© Prinetto – 2019

Disclaimer

- We disclaim any warranties or representations as to the accuracy or completeness of this material.
- Materials are provided “as is” without warranty of any kind, either express or implied, including without limitation, warranties of merchantability, fitness for a particular purpose, and non-infringement.
- Under no circumstances shall we be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this material.

Rel. 17.05.2019
© Prinetto – 2019

Goal

- This lecture presents the basic concepts of of ***Assembler Languages*** and the typical life cycle of assembler programs

Rel. 17.05.2019
© Prinetto – 2019

Prerequisites

- Module 0_4 : Digital Systems: Definitions and Taxonomies

Rel. 17.05.2019
© Prinetto – 2019

Homework

None

Rel. 17.05.2019

© Prinetto – 2019

Further readings

- **Students interested in making a reference to a text book on the arguments covered in this lecture can refer, for instance, to:**
  - ***G. Conte, A. Mazzeo, N. Mazzocca, P. Prinetto*: "Architettura dei calcolatori", *Città Studi, 2015* (Chapter 4 - Interfaccia di programmazione & Chapter 5 – Programmazione in linguaggio Assembler)**



Outline

- Introduction
- The program life cycle

Rel. 17.05.2019
© Prinetto – 2019

Outline

- Introduction
- The program life cycle

Rel. 17.05.2019
© Prinetto – 2019

Livello ISA

- **Con il termine *ISA - Instruction Set Architecture* si è soliti identificare tutti quegli aspetti dell'architettura di un sistema di elaborazione che sono visibili al *programmatore a livello assembler*.**

Rel. 17.05.2019
© Prinetto – 2019

Livello ISA

Questi comprendono:

- **i tipi di dati nativi**
- **le istruzioni**
- **i registri**
- **le modalità di indirizzamento**
- **l'architettura della memoria**
- **la gestione degli interrupt e delle eccezioni**
- **l'eventuale I/O esterno.**

Rel. 17.05.2019
© Prinetto – 2019

Linguaggi assembler

- **Ciascun processore ha il proprio linguaggio assembler (in termini di istruzioni macchina).**
- **Processori diversi appartenenti a una stessa famiglia possono riconoscere lo stesso linguaggio.**

Rel. 17.05.2019
© Prinetto – 2019

Caratteristiche di un linguaggio assembler

- Operazioni permesse
- Tipi di dato
- Formato delle istruzioni
- Registri utilizzabili
- Modi di indirizzamento
- Facilità d'uso
- ...

Rel. 17.05.2019
© Prinetto – 2019

Programmi Assembler

- **Sono composti di:**
  - ***istruzioni*: producono una istruzione macchina**
  - ***pseudo-istruzioni*: possono produrre più istruzioni macchina**
  - ***direttive*: comandi per l'assemblatore.**

Rel. 17.05.2019
© Prinetto – 2019

Programmi Assembler

- **Esempi:**
  - **Istruzione:** **ADD AX, 5**
  - **Direttiva:** **VAR1 DB ?**

Rel. 17.05.2019
© Prinetto – 2019

Formato delle Istruzioni nel Codice Sorgente

```
label: op_code_mnemonico operandi ;commento
```

dove

- `label`: **identifica l'indirizzo di partenza di una istruzione**
- `op_code_mnemonico`: **codice operativo**
- `operandi`: **in numero variabile da 0 a 3**
- `commento`: **qualsiasi sequenza di caratteri terminata da un fine-riga**

Esempio

```
lab1:   mov   ax, 5          ; carico 5 in ax
```


Rel. 17.05.2019
© Prinetto – 2019

Esempio di file sorgente

```
CR          EQU     13
            .MODEL large
            PUBLIC INPUT
            .DATA
ERR_MESS        DB      "Numero troppo grande", 0DH, 0AH, "$"
            .CODE
;*************************************************************************
;                       INPUT
; Procedura di lettura e conversione di un numero.
; Il numero letto e decodificato viene scritto in DX.
;*************************************************************************
INPUT   PROC    FAR
            PUSH    AX
            PUSH    BX
lab0:       XOR     DX, DX
lab1:       MOV     BX, 10
            MOV     AH, 1               ; legge un carattere
            INT     21H
```


© Prinetto – 2019

Esempio di file sorgente

```
        JNE     i_err
        MOV     DX, AX
        ADD     DX, BX          ; somma la cifra letta
        JC      i_err
        JMP     lab1
i_err:  LEA     DX, ERR_MESS
        MOV     AH, 9
        INT     21H
        JMP     lab0
fine:   POP     BX
        POP     AX
        RET
INPUT   ENDP
        END
```


Rel. 17.05.2019

© Prinetto – 2019

Formato delle Istruzioni nel Codice Oggetto

- **Ogni istruzione a livello di codice oggetto corrisponde a una sequenza di byte in numero variabile (tipicamente tra 1 e 6)**

Rel. 17.05.2019
© Prinetto – 2019

Formato delle Istruzioni macchina

- **Ciascuna istruzione macchina contiene, opportunamente codificate, le seguenti informazioni:**
  - **l'operazione che deve essere svolta (*codice operativo* o *opcode*)**
  - **gli eventuali *operandi* coinvolti nella operazione**



Rel. 17.05.2019
© Prinetto – 2019

Formato delle Istruzioni macchina

- **Ciascuna istruzione macchina contiene, opportunamente codificate, le seguenti informazioni:**
  - **l'operazione che deve essere svolta (*codice operativo* o *opcode*)**
  - **gli eventuali *operandi* coinvolti nella operazione**

in numero variabile tra 0 e 3



Rel. 17.05.2019
© Prinetto – 2019

Formato delle Istruzioni macchina

- **Ciascuna istruzione macchina contiene, opportunamente codificate, le seguenti informazioni:**
  - **l'operazione che deve essere svolta (*codice operativo* o *opcode*)**
  - **gli eventuali *operandi* coinvolti nella operazione**



il modo in cui si specifica ove risiede ciascun operando viene detto *modo di indirizzamento*

Rel. 17.05.2019
© Prinetto – 2019

Lunghezzza delle Istruzioni macchina

Le istruzioni possono avere:

- ***formato a lunghezza fissa*: tutte le istruzioni hanno la stessa lunghezza in numero di bit o di byte utilizzati (architetture RISC)**
- ***formato a lunghezza variabile*: utilizzano, a seconda dello specifico codice operativo, un numero di byte o di parole di memoria variabile. Ad esempio, da 1 a 6 byte nell'ISA Intel x86 (architetture CISC)**

Rel. 17.05.2019
© Prinetto – 2019

Variabili

- **Sono utilizzate per identificare in modo simbolico una zona di memoria, di cui può essere stato definito il tipo di dato relativo**
- **All'atto della definizione della variabile, si definiscono:**
  - **Il nome**
  - **La dimensione**
  - **Il tipo di dato contenuto (eventualmente)**
  - **Il valore di inizializzazione (eventualmente)**

Rel. 17.05.2019
© Prinetto – 2019

Identificatori

- **Sono i nomi che possono essere assegnati a etichette, variabili, procedure, costanti, segmenti.**
- **Sono così composti:**
  - **il primo carattere può essere una lettera (a-z, A-Z), oppure uno dei 4 caratteri @ _ $?**
  - **gli altri caratteri possono essere una lettera, un numero, o uno dei 4 caratteri sopra.**
- **Nel MASM 6.11 la lunghezza massima di un identificatore è 247 caratteri.**

© Prinetto – 2019

MASM 6.xx

- **Case insensitive**
- **Non è sensibile agli spazi**
- **Ogni istruzione deve occupare una riga: altrimenti la riga precedente deve terminare con **

Rel. 17.05.2019
© Prinetto – 2019

Costanti

- binarie: 001101B
- ottali: 15O, 15Q
- esadecimali: 0DH, 0BEACH (devono iniziare con un numero)
- decimali: 13, 13D
- stringhe: ‘S’, ‘Ciao’

Rel. 17.05.2019
© Prinetto – 2019

Operatori

- aritmetici: +, -, *, /, MOD
- logici: AND, OR, NOT, XOR
- relazionali: EQ, NE, LT, LE, GT, GE

Rel. 17.05.2019
© Prinetto – 2019

Esempio di file sorgente

```
CR          EQU     13
            .MODEL large
            PUBLIC INPUT
            .DATA
ERR_MESS    DB      "Numero troppo grande", 0DH, 0AH, "$"
            .CODE
;*************************************************************************
;                       INPUT
; Procedura di lettura e conversione di un numero.
; Il numero letto e decodificato viene scritto in DX.
;*************************************************************************
INPUT   PROC    FAR
        PUSH    AX
        PUSH    BX
lab0:   XOR     DX, DX
lab1:   MOV     BX, 10
        MOV     AH, 1               ; legge un carattere
        INT     21H
```


© Prinetto – 2019

Esempio di file sorgente

```
        JNE     i_err
        MOV     DX, AX
        ADD     DX, BX          ; somma la cifra letta
        JC      i_err
        JMP     lab1
i_err:  LEA     DX, ERR_MESS
        MOV     AH, 9
        INT     21H
        JMP     lab0
fine:   POP     BX
        POP     AX
        RET
INPUT   ENDP
        END
```


© Prinetto – 2019

Pseudo-Istruzioni

- **Sono direttive per l'Assemblatore, che non corrispondono a istruzioni macchina nel codice generato**
- **Principali categorie di pseudo-istruzioni:**
  - **definizione delle variabili**
  - **definizione di costanti**
  - **definizione dei segmenti**

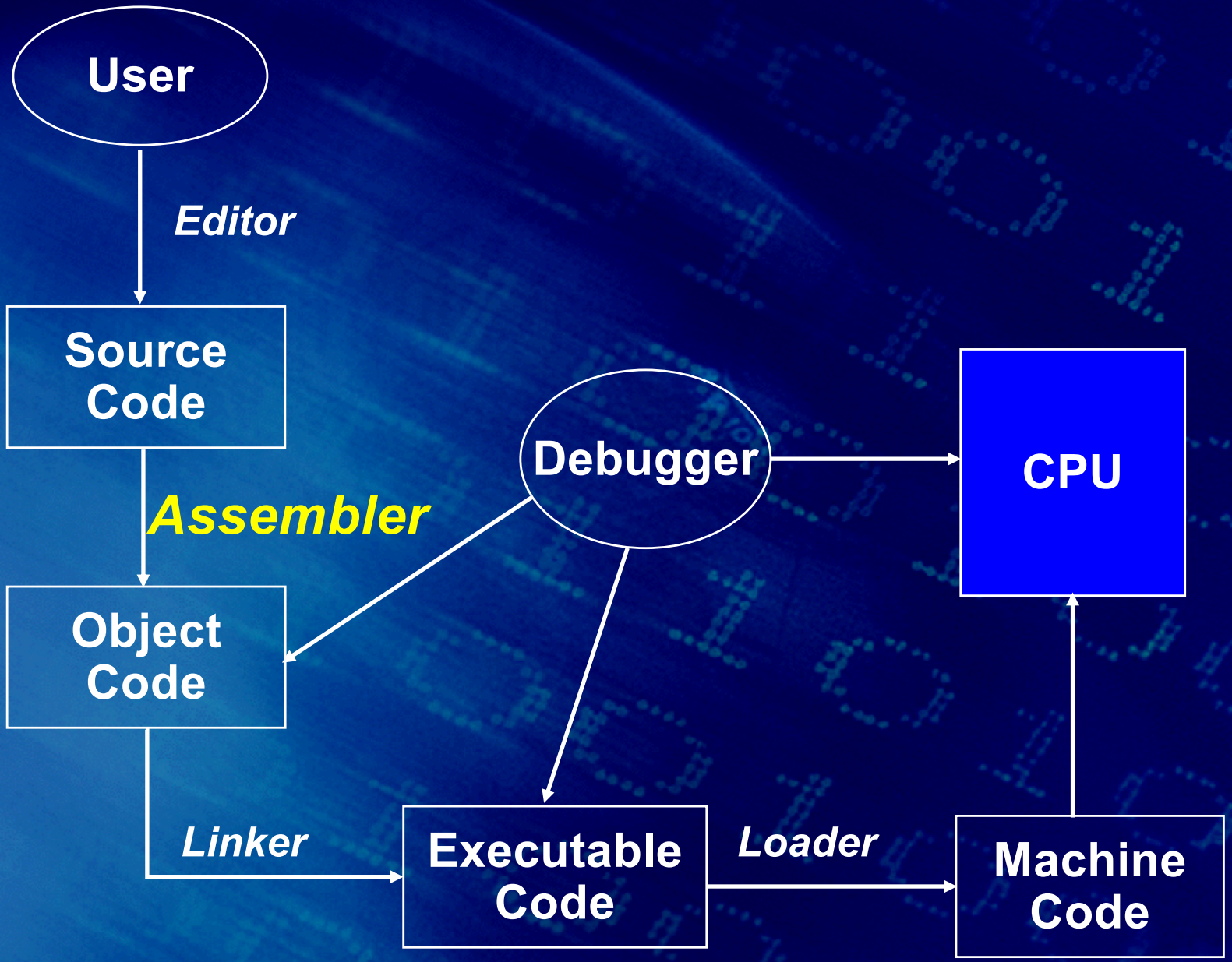
Rel. 17.05.2019
© Prinetto – 2019

Outline

- Introduction
- The program life cycle

Rel. 17.05.2019
© Prinetto – 2019

Program life cycle



Rel. 17.05.2019

© Prinetto – 2019

Assembler

- **Il file sorgente viene trasformato in file oggetto tramite:**
  - **Rimozione dei commenti**
  - **Associazione di ciascuna variabile simbolica ad una adeguata locazione di memoria**
  - **Traduzione in codice macchina di ciascuna istruzione.**

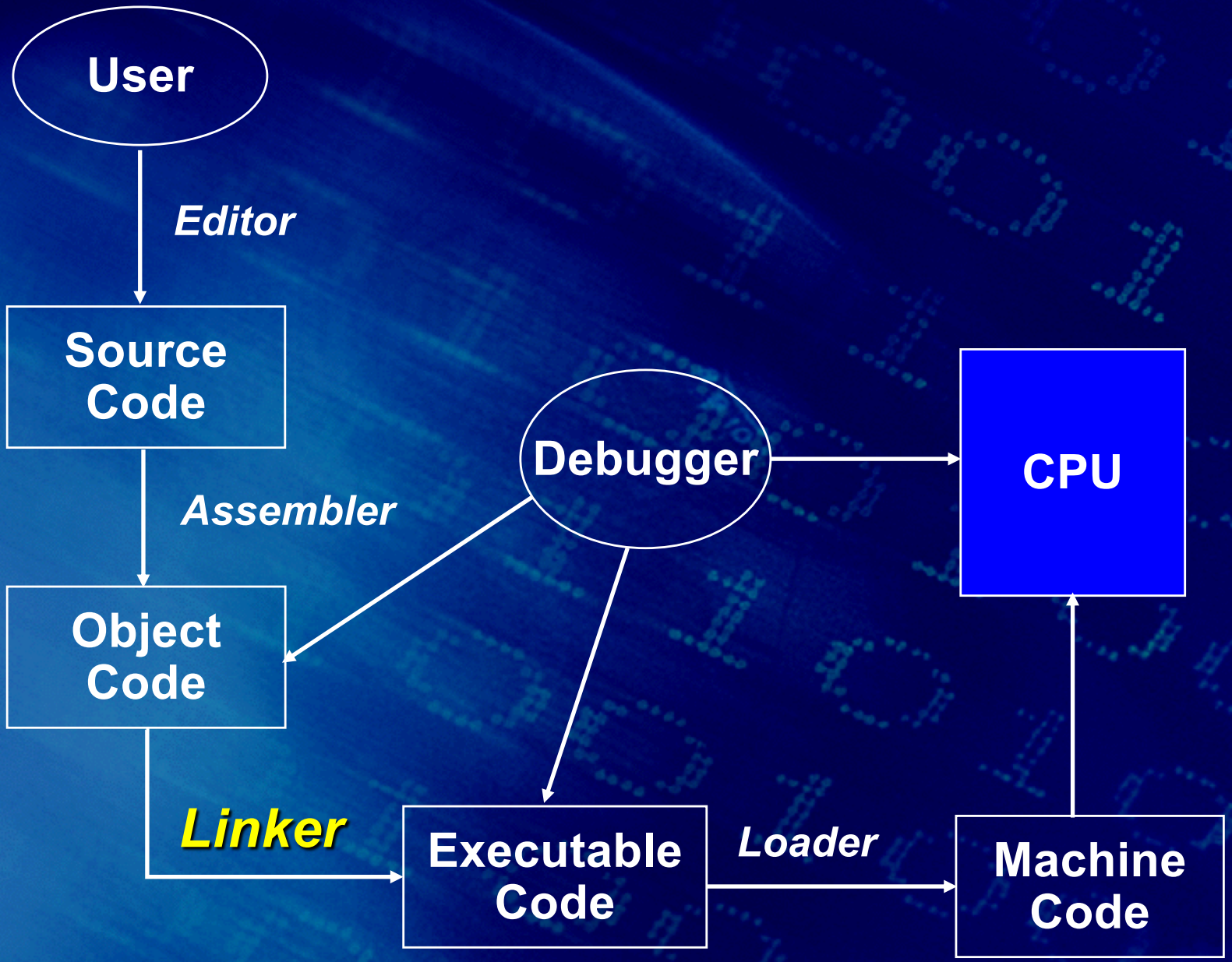
Rel. 17.05.2019
© Prinetto – 2019

Assembler

- **Alcune operazioni non possono ancora essere completate, ad esempio quelle che riguardano:**
  - **Le variabili definite in altri moduli**
  - **Le procedure definite in altri moduli.**

Rel. 17.05.2019
© Prinetto – 2019

Program life cycle



Rel. 17.05.2019

© Prinetto – 2019

Linker

- **Procede alla creazione del file eseguibile a partire da più file oggetto.**
- **Provvede a verificare la correttezza dei richiami a variabili e procedure definite in altri moduli e a generare gli indirizzi opportuni.**

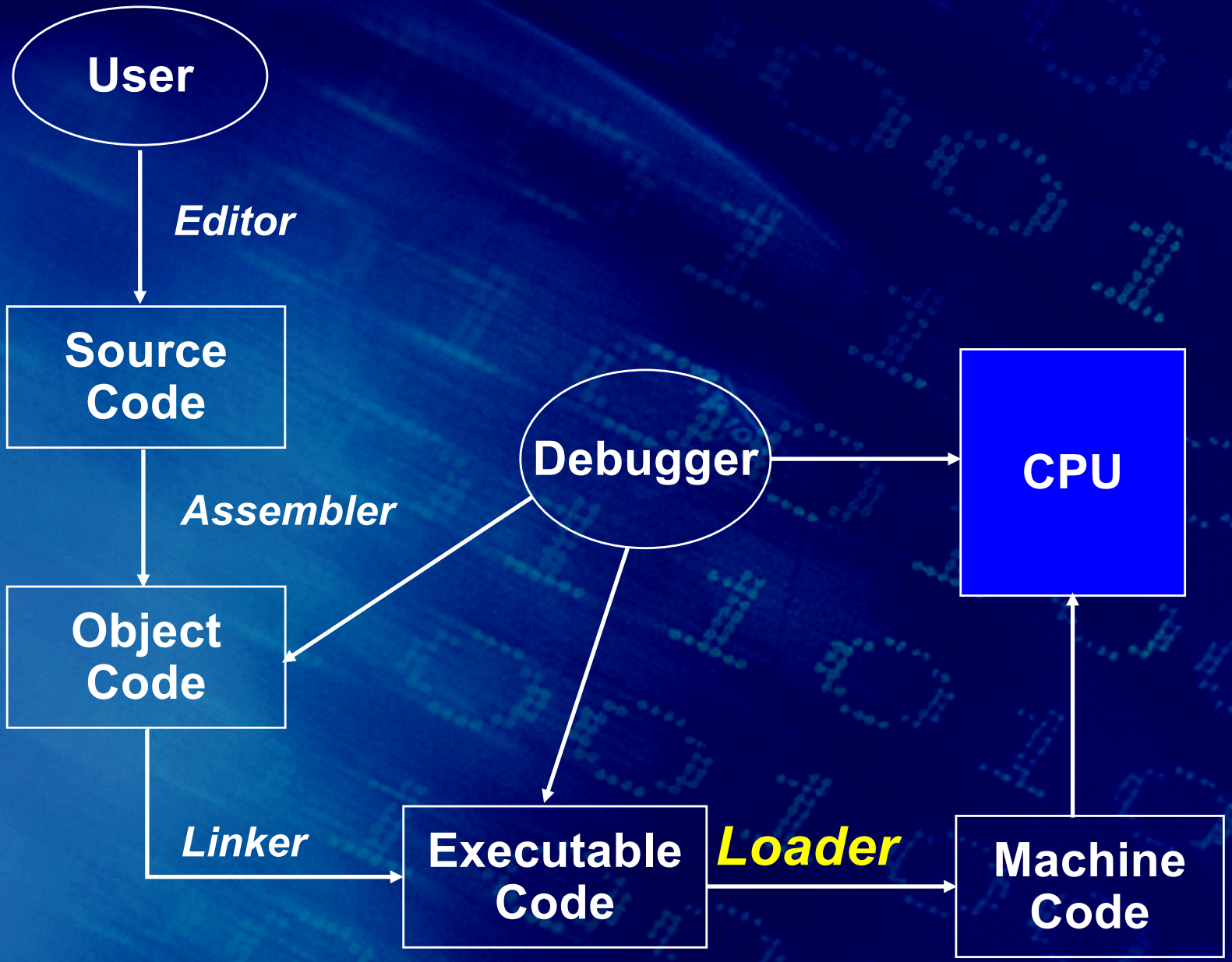
Rel. 17.05.2019
© Prinetto – 2019

Linker

- **Il file prodotto non è di solito ancora immediatamente eseguibile in quanto**
  - **Risiede su disco anziché in memoria**
  - **Deve essere indipendente dalla posizione in memoria in cui verrà caricato.**

Rel. 17.05.2019
© Prinetto – 2019

Program life cycle



Rel. 17.05.2019

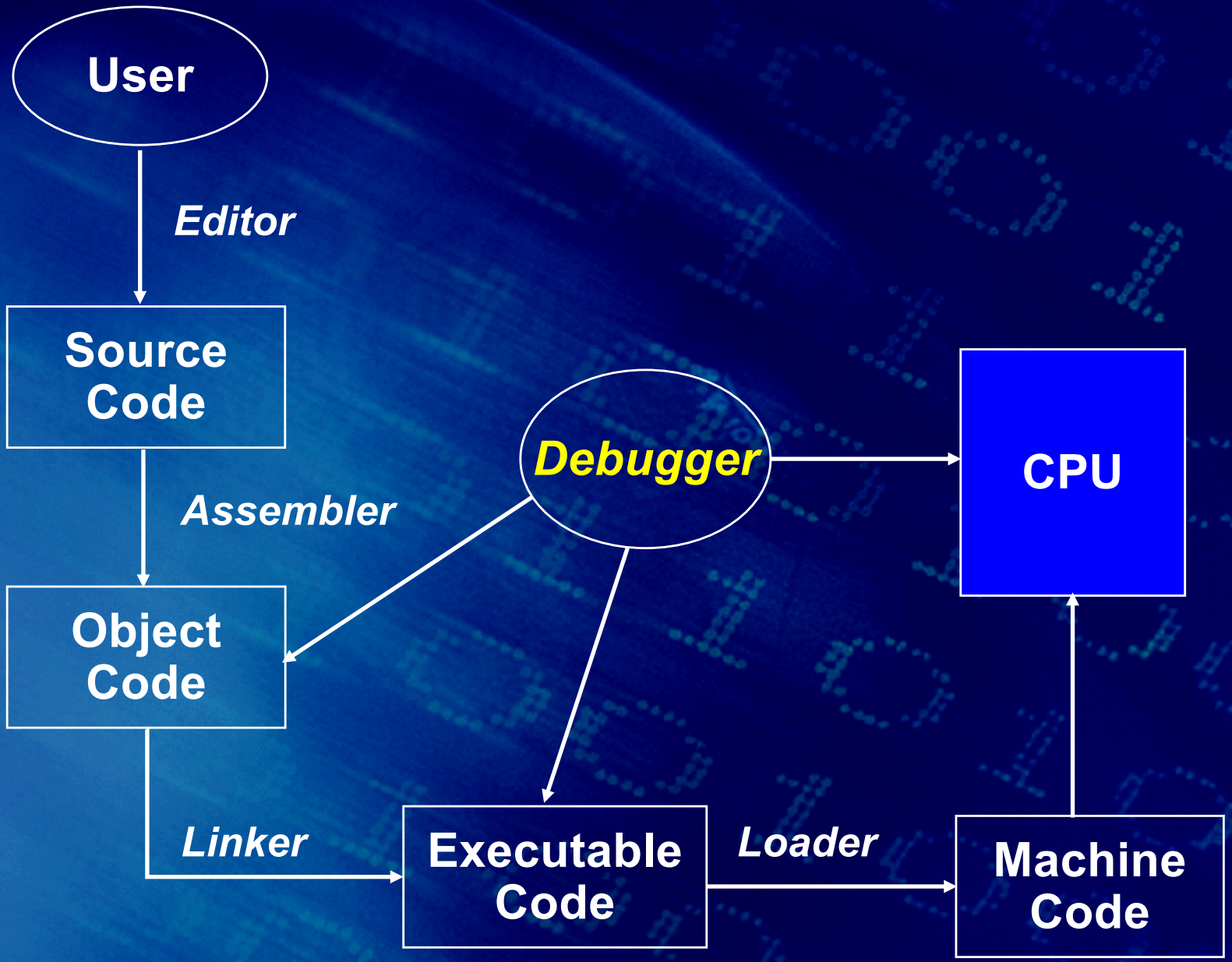
© Prinetto – 2019

Loader

- **Fa parte del Sistema Operativo.**
- **Provvede a**
  - **reperire il file eseguibile su disco**
  - **caricarlo in memoria, eseguendo le eventuali modifiche rese necessarie dopo che è stata decisa la sua posizione in memoria**
  - **fare in modo che il processore inizi l'esecuzione del programma.**

Rel. 17.05.2019
© Prinetto – 2019

Program life cycle



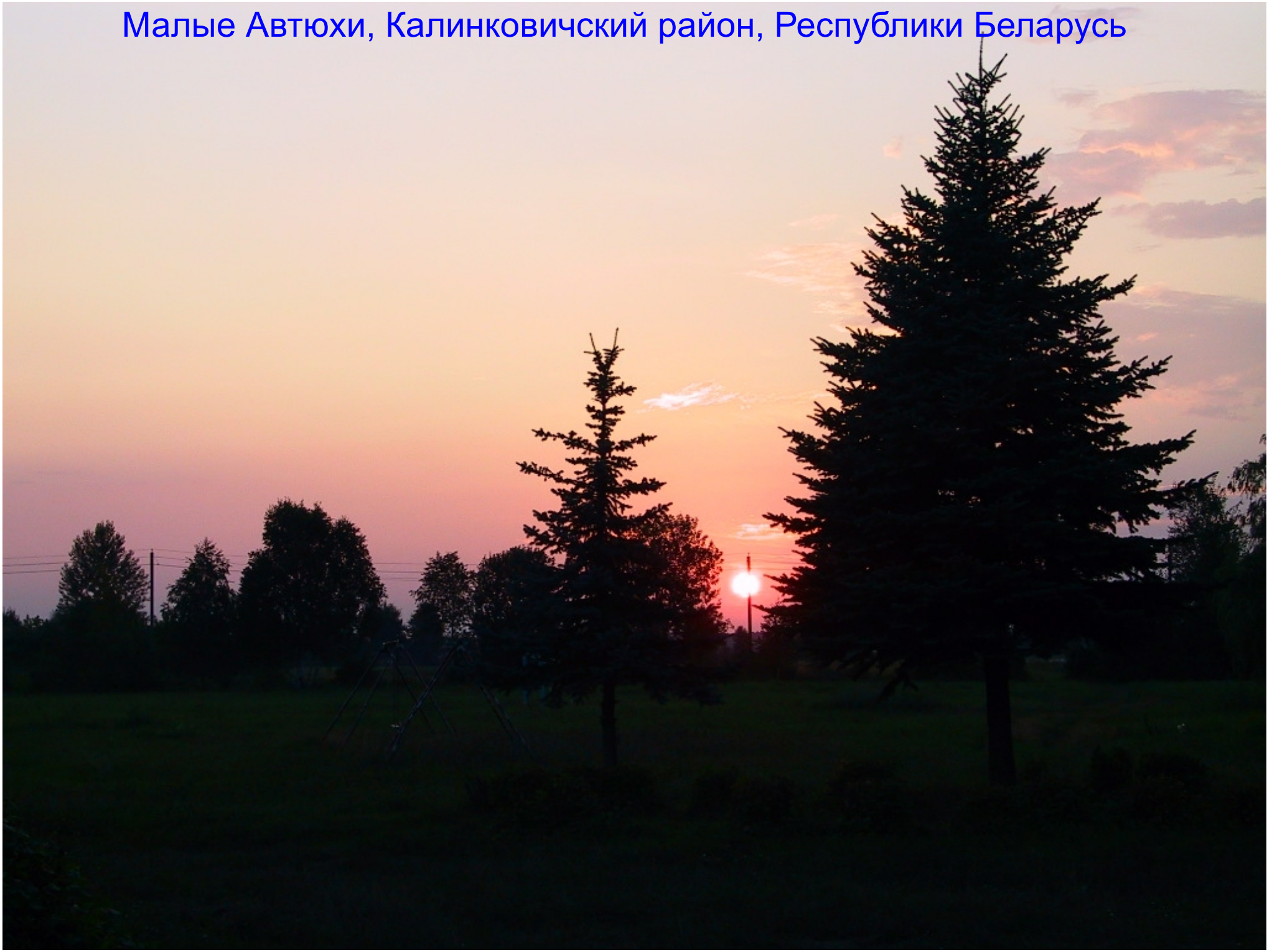
Rel. 17.05.2019

© Prinetto – 2019

Debugger

- **Interagisce con i processi di assemblaggio, link ed esecuzione, permettendo al programmatore di disporre di facilities quali:**
  - **Breakpoint**
  - **Esecuzione passo passo**
  - **Accesso a variabili e registri (in lettura e scrittura)**
  - **…**

Rel. 17.05.2019
© Prinetto – 2019


Малые Автюхи, Калинковичский район, Республики Беларусь