Programmazione I

Lezione 4

Recap: architetture e programmazione

- La **macchina** comprende un solo tipo di linguaggio, di tipo binario: l'attivazione o meno dei suoi circuiti

  (linguaggio macchina)

- Per renderlo leggibile, al linguaggio macchina è associato un linguaggio di **basso livello** con corrispondenza (sostanzialmente) 1 a 1

  (assembly)

- Esperimento "bottom up": possiamo creare nuove istruzioni componendo le precedenti (es. INC + J + JZ → ADD)
- Esperimento "top down": abbiamo poi strumenti automatici per la ri-traduzione (es. compilatore)

Il nostro primo esercizio di programmazione

```
1000: lw   $2, <2000>
1004: add  $3, $0, $0
1008: add  $4, $3, $0
1012: mult $4, $4, $4
1016: slt  $5, $4, $2
1020: beq  $5, $0, <1032>
1024: addi $3, $3, +1
1028: jump <1008>
1032: sw   $3, <3000>
```

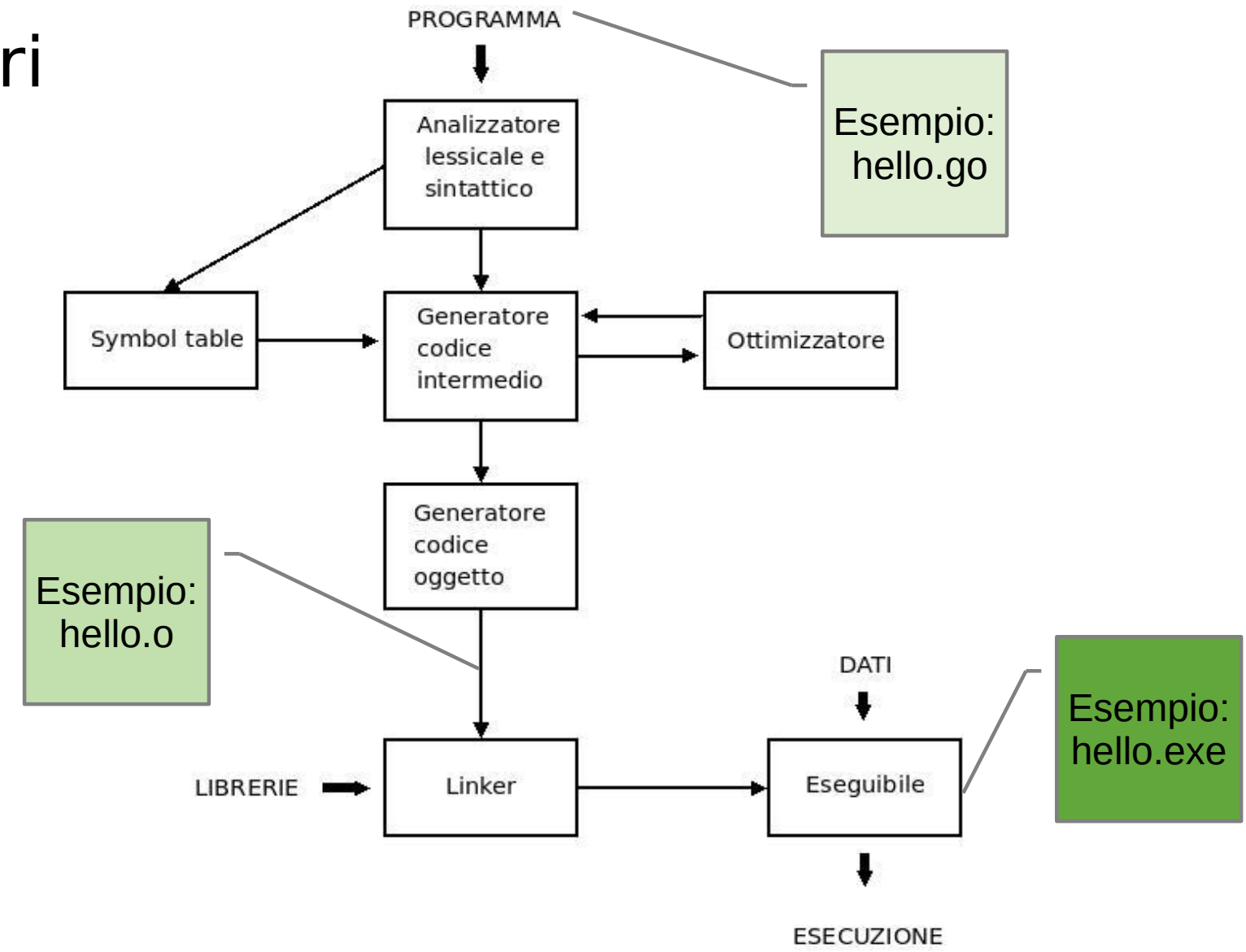
Recap: programmazione ed architetture

- Organizzare la complessità di un problema reale utilizzando istruzioni assembly è **teoricamente possibile**, ma **praticamente impercorribile**.

- Procedimento “top down”

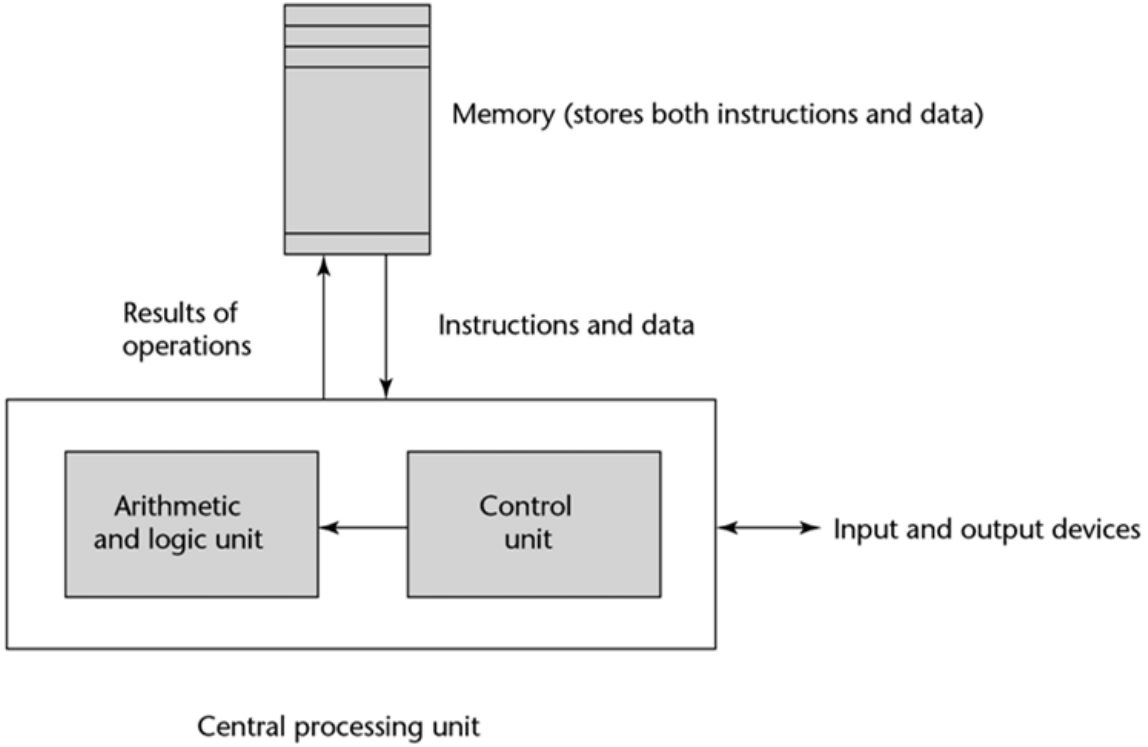
- Ci affidiamo a linguaggi ad **alto livello**, che poi sono tradotti in modo automatico in linguaggi a più basso livello

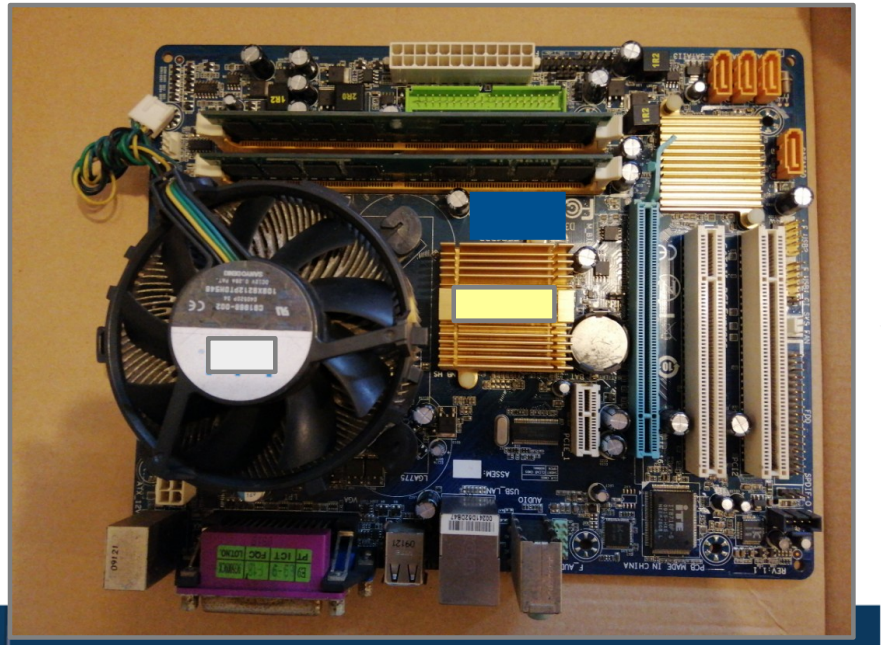
Compilatori



Macchina di Von Neumann

- **Memoria: stato** della macchina
  (i.e. uno snapshot: se ripristinassi tutto ritornerei esattamente alla stessa condizione)
- **CPU:** comandata con **istruzioni**, da eseguire in sequenza





Esecuzione di un programma

- Osservazione: dove sono memorizzati file sorgente, file oggetto, eseguibile (ma anche dati)?
- Fatto: secondo l'architettura di Von Neumann, codice e dati devono essere conservati in RAM
  - Istruzioni:
    - eseguite con ciclo FETCH → DECODE → EXECUTE
  - Dati:
    - portati da RAM ai registri (e viceversa)

Esecuzione di un programma

- Osservazione: dove sono memorizzati file sorgente, file oggetto, eseguibile (ma anche dati)?
- Soluzione: un eseguibile è
  - conservato su disco
  - caricato in RAM al momento dell'avvio (**loading time**)
  - avviato dalla RAM (**run time**)

Dal sorgente all’eseguibile

Per ottenere un’applicazione:

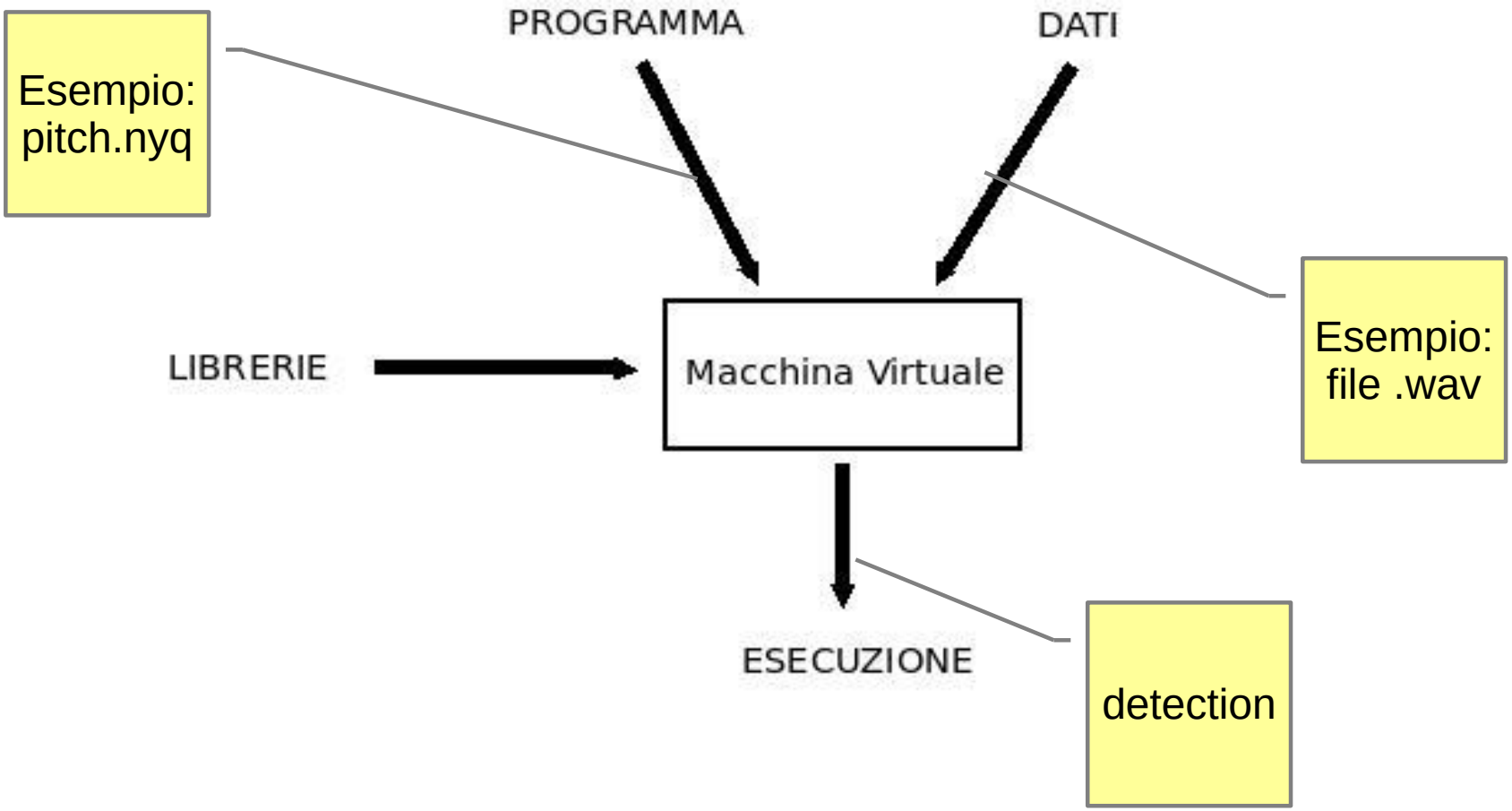
- Editing codice sorgente (programmazione vera e propria)
- Creazione di codice oggetto **(compile time)**
- Creazione eseguibile **(linking time)**
- Esecuzione **(run time)**

Esempio: hello.go

Esempio: visualizzazione assembly hello.o

Esempio: esecuzione hello.exe

Interpreti


Esempio:
pitch.nyq
PROGRAMMA
DATI
LIBRERIE
Macchina Virtuale
Esempio:
file .wav
ESECUZIONE
detection

Dal sorgente all’*esecuzione*

Da terminale ...

- Editing del codice sorgente e run simultaneo

Esempio: hello.py

Per chi è curioso sulla Turing-equivalenza

- Equivalenza per “emulazione” (es. da assembly a C)

  http://spimsimulator.sourceforge.net/

- … non per forza “decompilazione” (anche se qualche tentativo si può anche fare)!

  https://www.hex-rays.com/products/decompiler/

  http://derevenets.com/

- … e se i linguaggi sono simili anche i tentativi di traduzione pura possono essere sorprendenti!

  https://www.mtsystems.com/

Data abstraction

```
1000: lw   $2, <2000>
1004: add  $3, $0, $0
1008: add  $4, $3, $0
1012: mult $4, $4, $4
1016: slt  $5, $4, $2
1020: beq  $5, $0, <1032>
1024: addi $3, $3, +1
1028: jump <1008>
1032: sw   $3, <3000>
```

L’idea di variabile

- Astrazione di una (o più) celle di memoria
- Esempio: uso di variabili in un ambiente interattivo (python)
  - Il concetto di assegnamento: valori a variabili
  - Lettura di valori in variabili
  - Ancora assegnamento: variabili a variabili
  - Espressioni con variabili
  - Variabili e numeri (interi, floating)