Macchine Astratte

Luca Abeni

February 22, 2017

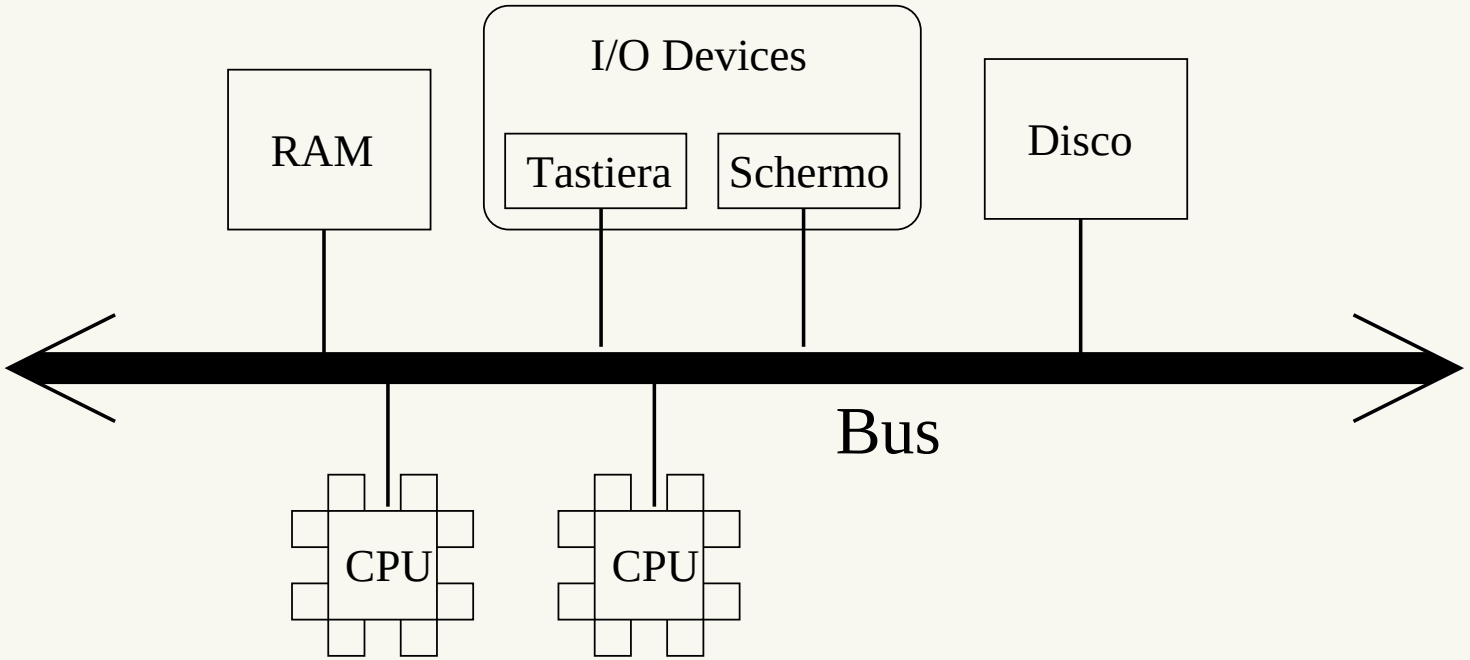
Architettura dei Calcolatori - 1

- Un computer è composto almeno da:
  - Un processore (CPU)
    - Esegue le istruzioni macchina
    - Per fare questo, può muovere dati da/verso la memoria
  - Una memoria centrale (RAM)
    - Memorizza dati e programmi (sequenze di istruzioni macchina)
    - Veloce, ma volatile
  - Una Memoria di massa
    - Più lenta della RAM, ma persistente
  - Alcune periferiche di ingresso/uscita (I/O device)

Architettura dei Calcolatori - 2

- I vari componenti di un computer (una o più CPU, RAM, I/O device, ...) sono connessi fra loro tramite un bus
  - Esempio: bus di sistema
  - Composto da una serie di collegamenti elettrici
- Usato per trasmettere istruzioni macchina e dati fra CPU e RAM
- Usato per Input o Output di dati dai device da memoria di massa

Architettura di Von Neumann



- Memoria che contiene sia dati che istruzioni
- Bus che collega CPU a memoria e dispositivi di ingresso / uscita

Il Processore

- Preleva le istruzioni macchina dalla memoria e le esegue
  - Esecuzione: può aver bisogno di modificare locazioni di memoria (leggere / scrivere dati)
- Parte operativa e parte di controllo
  - Parte di controllo: preleva ed esegue le istruzioni
  - Parte operativa (Arithmetic Logic Unit - ALU): esegue le istruzioni (aritmetiche e logiche)
  - CPU moderne hanno anche una parte floating point (FPU) ed altro...
- Contiene internamente dei registri
  - Visibili od invisibili al programmatore

Registri del Processore

- Invisibili (non acceduti direttamente da istruzioni macchina):
  - Address Register (AR): indirizzo da accedere sul bus
  - Data Register (DR): dato da scrivere / dato letto
- Visibili (menzionati dalle istruzioni macchina):
  - Program Counter (PC): indirizzo della prossima istruzione macchina da eseguire (AKA IP - Instruction Pointer)
  - Status Register (SR): insieme di flag relativi al risultato di operazioni ALU o allo stato della macchina (AKA F - Flag register)
  - Alcuni registri dati ed indirizzi

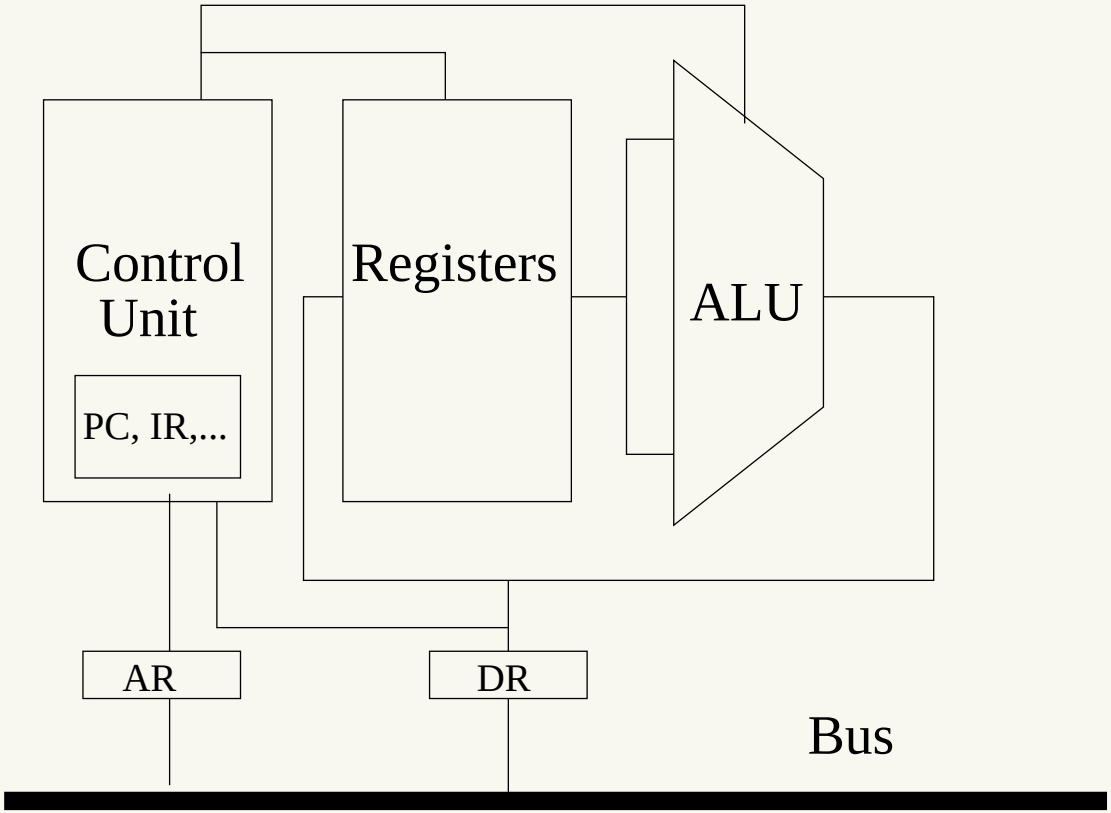
Esecuzione di un'Istruzione

- Lettura istruzione da eseguire (fetch)
  - Copia PC in AR
  - Trasferisci dato (indirizzato da AR) da RAM a DR
  - Salva DR in un registro invisibile
  - Incrementa PC
- Decodifica istruzione: interpreta l'istruzione
- Esecuzione istruzione: esegui l'istruzione decodificata
  - Se è necessario caricare dati da memoria, setta AR, leggi DR, etc...
  - Se è necessario salvare dati in memoria, setta AR, scrivi DR, etc...
  - Si può modificare PC (jump, etc...)

La Memoria Centrale

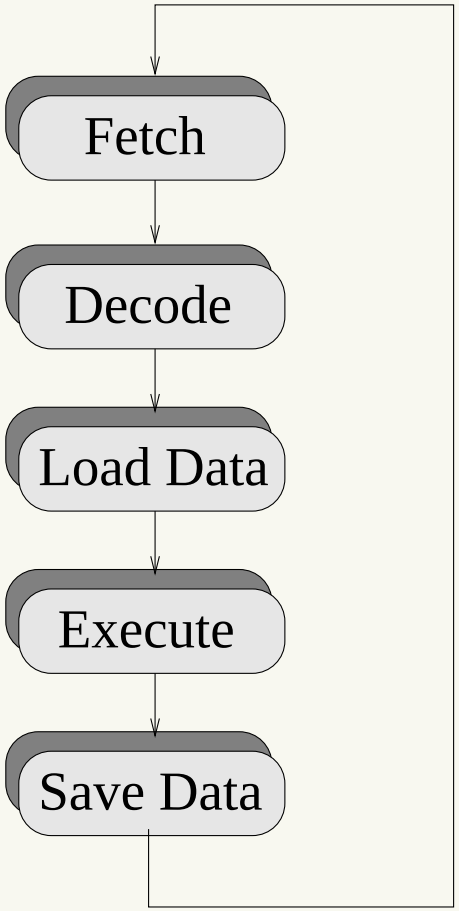
- Modello di Von Neumann $\rightarrow$ la stessa memoria può contenere dati o istruzioni
- Acceduta tramite bus
- Insieme di celle (o locazioni) di 8 bit l'una
- Accesso alla memoria:
  - Caricamento indirizzo da accedere nel registro AR
  - In caso di scrittura, caricamento del dato da scrivere nel registro DR
  - Segnalazione (sul bus) dell'operazione da eseguire (lettura / scrittura)
  - in caso di lettura, il dato letto si trova ora in DR

Esempio di CPU



- Esempio chiaramente semplificato
- CPU moderne: struttura più complicata
  - Pipeline
  - Parallelizzazione
  - ...

Esecuzione di un Programma



- CPU: esegue ciclicamente sempre le stesse operazioni
  - Esecuzione prevalentemente sequenziale di istruzioni macchina
- Macchina progettata per eseguire un proprio linguaggio
  - Linguaggio Macchina (LM)

Macchine Fisiche...

- Computer: macchina (fisica) progettata per eseguire programmi
- Ogni macchina esegue programmi scritti in un suo linguaggio
- Stretta relazione fra macchina e linguaggio
  - Ogni macchina ha un proprio linguaggio che è in grado di capire ed eseguire
  - Lo stesso linguaggio può essere capito da più macchine diverse
- Esecuzione di un programma: ciclo (infinito) fetch/decode/load/execute/save
  - CPU: implementazione in hw del ciclo
  - Algoritmo (interprete) che è in grado di capire ed eseguire un suo linguaggio macchina

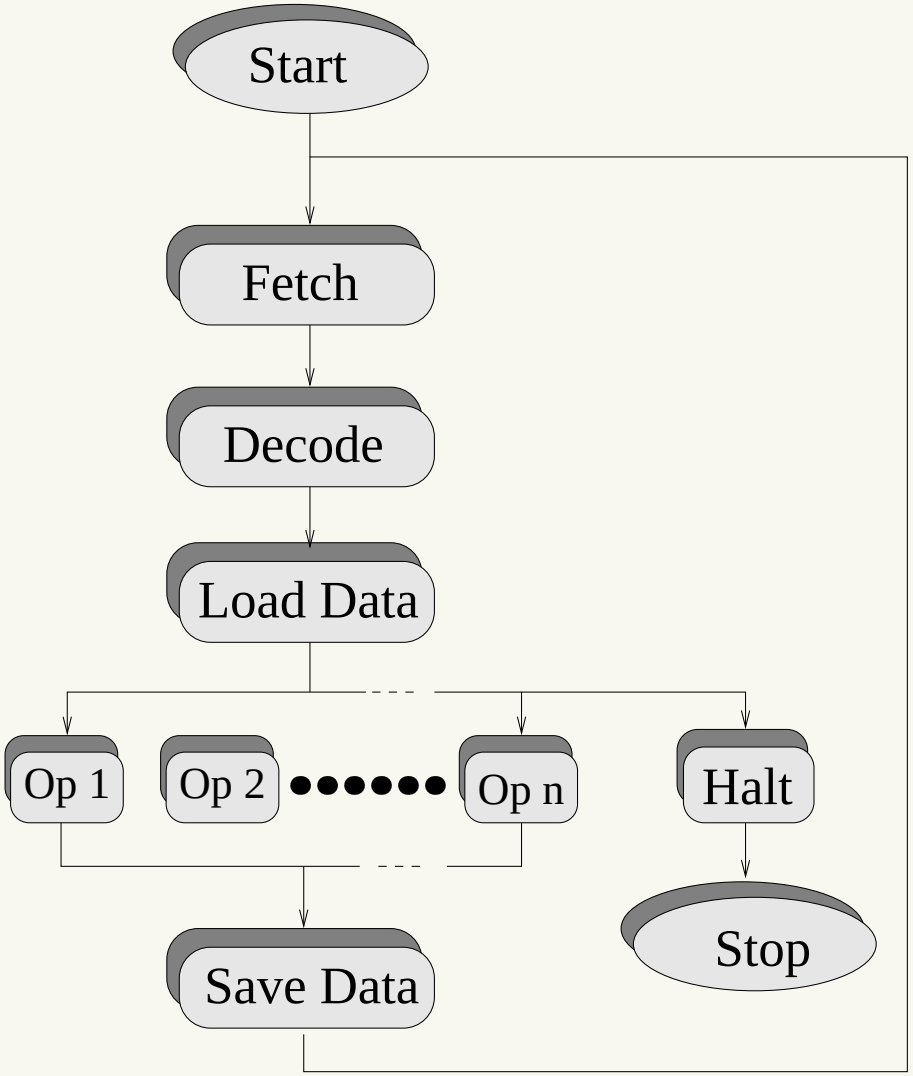
...E Macchine Astratte!

- L'algoritmo che esegue un programma non deve necessariamente essere implementato in hw...
- Implementazione software: Macchina Astratta
  - Insieme di algoritmi e strutture dati che permettono di memorizzare ed eseguire programmi
- Come per macchine fisiche (CPU), ad ogni macchina astratta è associato un suo linguaggio
  - $\mathcal{M}_{\mathcal{L}}$: macchina astratta che capisce ed esegue il linguaggio $\mathcal{L}$
  - $\mathcal{L}$ è il *linguaggio macchina* di $\mathcal{M}_{\mathcal{L}}$
  - Programma: sequenza di istruzioni in $\mathcal{L}$
- $\mathcal{M}_{\mathcal{L}}$ è un possibile modo per descrivere $\mathcal{L}$

Funzionamento di una Macchina Astratta

- Per eseguire un programma scritto in $\mathcal{L}$, $\mathcal{M}_{\mathcal{L}}$ deve:
  1. Eseguire operazioni elementari
     - In hw, ALU
  2. Controllare il flusso di esecuzione
     - Esecuzione non solo sequenziale (salti, cicli, etc...)
     - In hw, gestione PC
  3. Trasferire dati da / a memoria
     - Modalità di indirizzamento, ...
  4. Gestire la memoria
     - Allocazione dinamica, gestione stack, memoria dati / programmi, etc...

Esempio di Macchina Astratta



- Ciclo di esecuzione: simile a CPU...
- ... Ma implementato in sw!

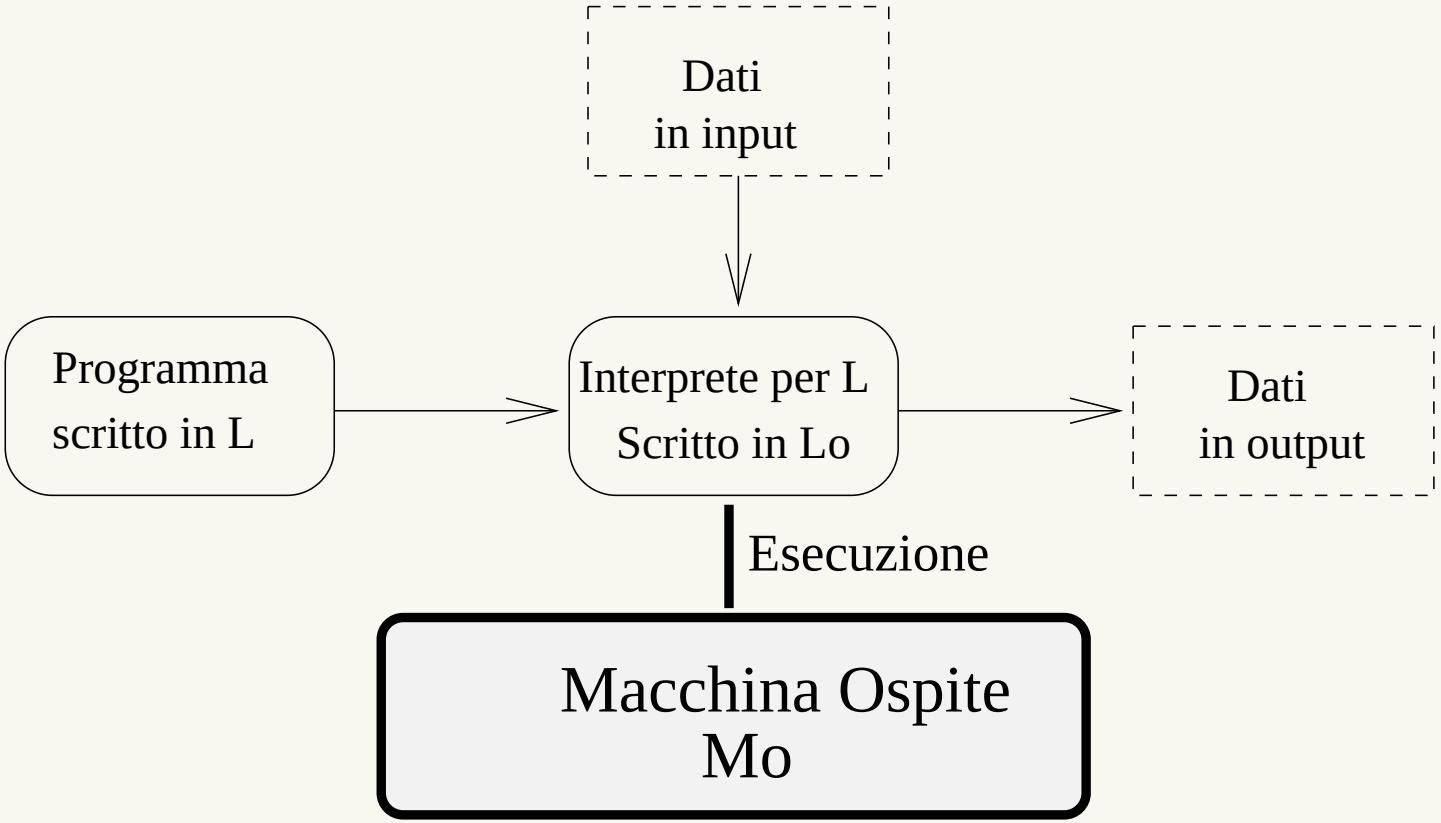
Implementazione di un Linguaggio

- $\mathcal{M}_{\mathcal{L}}$ capisce il suo linguaggio macchina $\mathcal{L}$
  - Un solo linguaggio macchina per macchina astratta
- $\mathcal{L}$ è capito/eseguito da varie (infinite) macchine astratte
  - Possono differire per implementazione, strutture dati, etc...
- Implementazione di un linguaggio $\mathcal{L}$: realizzazione di una macchina astratta $\mathcal{M}_{\mathcal{L}}$ in grado di eseguire programmi scritti in tale linguaggio
  - Implementazione hw, sw o firmware

Implementazione Software

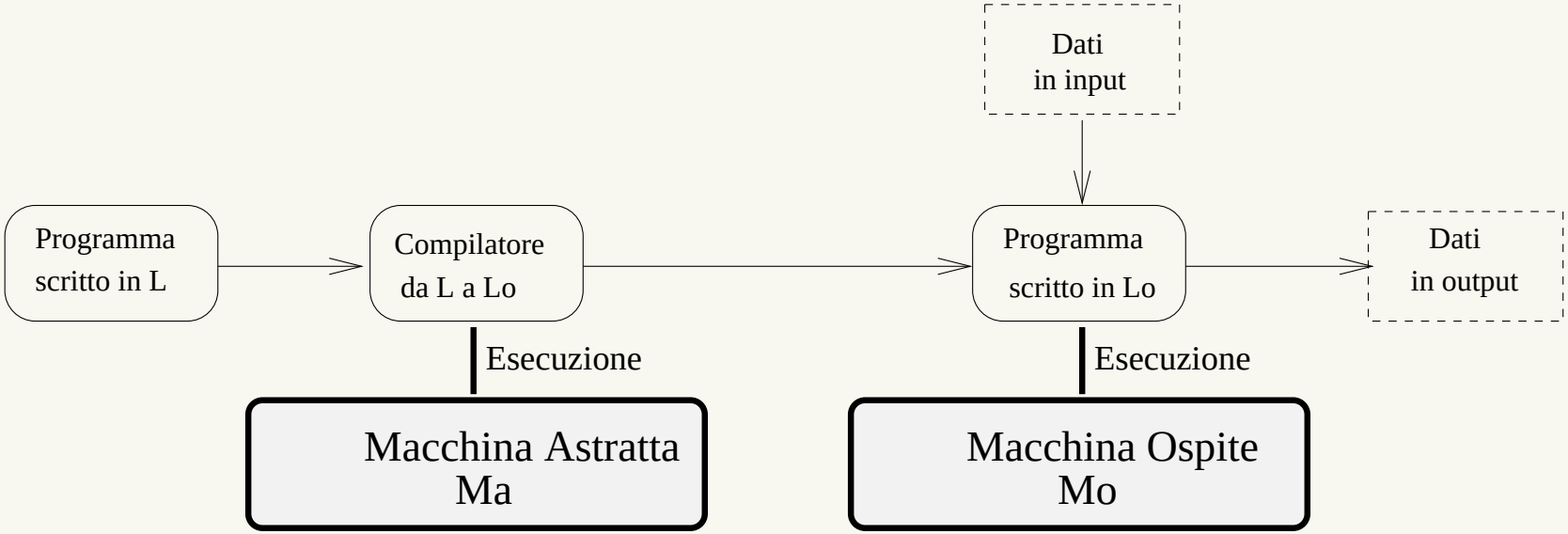
- Implementiamo in software $\mathcal{M}_{\mathcal{L}}$ (esegue programmi scritti in $\mathcal{L}$)
- Il software esegue su una Macchina Ospite $\mathcal{M}o_{\mathcal{L}o}$ (con linguaggio macchina $\mathcal{L}o$)
- Due tipologie di soluzione: *interpretativa* e *compilativa*
  - Interprete: programma scritto in $\mathcal{L}o$ che capisce ed esegue $\mathcal{L}$
    - Implementa il ciclo fetch/decode/load/exec/save
  - Compilatore: programma che capisce traduce altri programmi da $\mathcal{L}$ a $\mathcal{L}o$

Implementazione Puramente Interpretativa



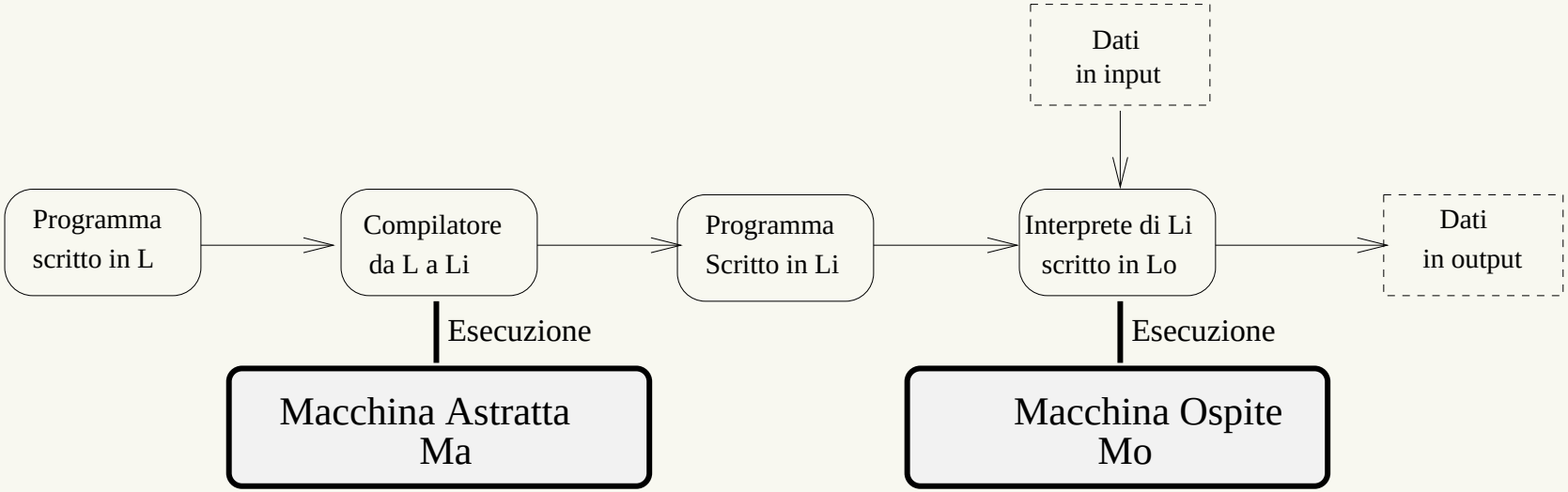
- Interprete: programma scritto in $\mathcal{L}o$ (esegue su $\mathcal{M}o_{\mathcal{L}o}$) che capisce ed esegue programmi scritti in $\mathcal{L}$
- Traduzione da $\mathcal{L}o$ a $\mathcal{L}$ “istruzione per istruzione”

Implementazione Puramente Compilativa



- Traduzione dell'intero programma da $\mathcal{L}$ a $\mathcal{L}o$ *prima di eseguirlo*
- Traduzione è effettuata da un apposito programma, il Compilatore
  - Compilatore: non necessariamente scritto in $\mathcal{L}o$
  - Può eseguire su una macchina astratta $\mathcal{M}a$ diversa da $\mathcal{M}o_{\mathcal{L}o}$
    - Cross-compilazione!

Implementazione “Ibrida”



- Implementazione non puramente compilativa ne’ puramente interpretativa
- Compilatore traduce in un *linguaggio intermedip* $\mathcal{L}i$
- Interprete (o altro) esegue su $\mathcal{M}o_{\mathcal{L}o}$ programma scritto in $\mathcal{L}i$
  - Java: compilatore $\rightarrow$ bytecode, poi JVM
  - C: compilatore generalmente genera codice che ha bisogno di SO e runtime per eseguire

Implementazioni Ibride - 2

- Dipendentemente dalle differenze e similitudini fra $\mathcal{L}i$ ed $\mathcal{L}o$ si parla di implementazioni prevalentemente compilative o interpretative
  - Java: $\mathcal{L}i ==$ bytecode $\rightarrow$ molto diverso da linguaggio macchina
    - Implementazione prevalentemente interpretativa
    - Ma alcune JVM sono basate su compilatori!!!
  - C: $\mathcal{L}i \approx$ linguaggio macchina
    - Ma non proprio identico: contiene chiamate a system call, funzioni di libreria, ...
    - Eccezione: compilatori “freestanding” (usati per compilare kernel)

Compilatori vs Interpreti

- Meglio compilare o interpretare? Dipende...
- Implementazione puramente interpretativa:
  - Implementazione di $\mathcal{M}_{\mathcal{L}}$ poco efficiente...
  - ... Ma più flessibile e portabile!
  - Debugging semplificato (interpretazione a run-time)
- Implementazione puramente compilativa:
  - Implementazione di $\mathcal{M}_{\mathcal{L}}$ più efficiente...
  - ... Ma anche più complessa ("distanza" fra $\mathcal{L}$ e $\mathcal{L}_i/\mathcal{L}_o$!
  - Debugging generalmente più complesso
- In medio stat virtus...

In Pratica...

- Come detto, è talvolta difficile stabilire se un linguaggio è *prevalentemente* compilato o interpretato...
- Comunque, proviamo
  - Linguaggi tipicamente implementati in modo compilativo:
    - C, C++, FORTRAN, Pascal, ADA
  - Linguaggi tipicamente implementati in modo interpretativo:
    - LISP, ML, Perl, Postscript, Pascal, Prolog, Smalltalk, Java
- Siamo così sicuri riguardo a Java? E LISP?

Implementazione in Firmware

- Abbiamo visto come implementare macchine astratte in sw...
- ... Mentre una CPU è un'implementazione hw
- Esistono anche interpretazioni "intermedie"
  - Esempio: CPU microprogrammate
  - La macchina astratta che esegue il linguaggio macchina della CPU non è implementata in hw...
  - ... Ma è implementata da un microinterprete
  - Ciclo fetch/decode/load/execute/save implementato usando *microistruzioni* invisibili a normali utenti
  - Strutture dati e algoritmi MA realizzati da microprogrammi
- Alta velocitá, flessibilitá maggiore che hw puro

Definizioni

- Definizione più formale di interpreti e compilatori...
- Prima di tutto, programma come funzione
  - Programma scritto in linguaggio $\mathcal{L}$:

$$P^{\mathcal{L}} : \mathcal{D} \rightarrow \mathcal{D}$$

    - $\mathcal{D}$: insieme dei dati di input ed output del programma
  - $P^{\mathcal{L}}(i) = o$
    - $i \in \mathcal{D}$: input; $o \in \mathcal{D}$: output
- Interpreti e compilatori sono anche loro programmi...
  - Funzioni “un po’ particolari”, che ricevono in input altri programmi

Definizione di Interprete

- Interprete per linguaggio $\mathcal{L}$ scritto in linguaggio $\mathcal{L}o$
  - Implementa macchina astratta $\mathcal{M}_{\mathcal{L}}$ su macchina astratta $\mathcal{M}o_{\mathcal{L}o}$
- Funzione $\mathcal{I}_{\mathcal{L}}^{\mathcal{L}o} : (\mathcal{P}r^{\mathcal{L}} \times \mathcal{D}) \rightarrow \mathcal{D}$
  - $\mathcal{P}r^{\mathcal{L}}$: insieme dei programmi $P^{\mathcal{L}}$ scritti in linguaggio $\mathcal{L}$
- $\forall i \in \mathcal{D}, \mathcal{I}_{\mathcal{L}}^{\mathcal{L}o}(P^{\mathcal{L}}, i) = P^{\mathcal{L}}(i)$
  - Per qualsiasi input i, l'interprete applicato a $P^{\mathcal{L}}$ ed i ritorna lo stesso risultato ritornato da $P^{\mathcal{L}}$ applicato ad i
  - $P^{\mathcal{L}}(i)$ è calcolata dalla macchina astratta $\mathcal{M}_{\mathcal{L}}$, mentre $\mathcal{I}_{\mathcal{L}}^{\mathcal{L}o}(P^{\mathcal{L}}, i)$ è calcolata dalla macchina astratta ospite $\mathcal{M}o_{\mathcal{L}o}$

Definizione di Compilatore

- Compilatore da linguaggio $\mathcal{L}$ a linguaggio $\mathcal{L}o$ (scritto in linguaggio $\mathcal{L}a$)
  - Implementa $\mathcal{M}_{\mathcal{L}}$ su $\mathcal{M}o_{\mathcal{L}o}$...
  - ...trasformando programmi $P^{\mathcal{L}} \in \mathcal{P}r^{\mathcal{L}}$ in programmi $P^{\mathcal{L}o} \in \mathcal{P}r^{\mathcal{L}o}$
- Funzione $\mathcal{C}^{\mathcal{L}a}_{\mathcal{L},\mathcal{L}o} : \mathcal{P}r^{\mathcal{L}} \rightarrow \mathcal{P}r^{\mathcal{L}o}$
- $\forall i \in \mathcal{D}, P^{\mathcal{L}o}_C = \mathcal{C}^{\mathcal{L}a}_{\mathcal{L},\mathcal{L}o}(P^{\mathcal{L}}) \Rightarrow P^{\mathcal{L}o}_C(i) = P^{\mathcal{L}}(i)$
  - Per qualsiasi input i, il programma compilato $P^{\mathcal{L}o}_C = \mathcal{C}^{\mathcal{L}a}_{\mathcal{L},\mathcal{L}o}(P^{\mathcal{L}})$ applicato ad i ritorna lo stesso risultato ritornato da $P^{\mathcal{L}}$ applicato ad i
  - $P^{\mathcal{L}}(i)$: calcolata dalla macchina astratta $\mathcal{M}_{\mathcal{L}}$; $P^{\mathcal{L}o}_C$: calcolata dalla macchina ospite $\mathcal{M}o_{\mathcal{L}o}$
  - Compilatore $\mathcal{C}^{\mathcal{L}a}_{\mathcal{L},\mathcal{L}o}(P^{\mathcal{L}})$: calcolata da $\mathcal{M}a$