Note su
Macchine Astratte*

Sommario

Nelle presenti note, partendo dalla definizione del modello computazionale delle Macchine di Turing, si arriva alla nozioni di Macchina Astratta[1], realizzazione di macchine astratte e sistemi a livelli di macchine astratte. Queste ultime nozioni possono essere utilizzate come riferimento generale per inquadrare la maggior parte degli argomenti che si affronteranno nel corso di Architettura degli elaboratori

*Le presenti note sono parzialmente basate sulle note del prof. Levi dell'Universita' di Pisa. LaTeX editing by Andrea Maugeri.

[1]Al posto del termine "Macchina Astratta" spesso si utilizza quello di "Macchina Vituale (Virtual Machine)".

Indice

1 Cosa vuol dire computare?

Nonostante possa sembrare banale, è tutt'altro che facile dare una definizione allo stesso tempo formale e generale del concetto di **computazione**.
Possiamo dire che computare significa ridurre una informazione da una forma implicita ad una forma esplicita in modo effettivo, ovvero rielaborare un insieme di "informazioni" al fine di presentarli in una forma di più diretta comprensione.
L'approccio classico alla formalizzazione del concetto di calcolo è fondato sull'idea di fornire delle regole meccanizzabili, che esprimano il procedimento necessario all'esplicitazione dell'informazione iniziale in modo effettivo. Nel processo di calcolo entrano quindi in gioco entità come:

- dati
- passo elementare di computazione (effettivo, meccanizzabile, discreto)

Diversi modi di formalizzare questi elementi portano a diversi **modelli computazionali**. Un modello computazionale è un formalismo matematico in cui viene rappresentato in modo astratto, ma preciso, un particolare modo di intendere in concetto di input, di output, di passo elementare di computazione e di come organizzare un insieme di passi elementari di computazione in una descrizione di un procedimento effettivo (algoritmo) che permetta di rendere esplicito il contenuto informativo all'inizio presente solo in forma implicita.
È da sottolineare come il concetto generale di **computazione** possa essere formalizzato in vari modi, dando origine a differenti modelli computazionali, ognuno dei quali utilizza particolati nozioni specifiche per rappresentare i concetti più generali.
I modelli computazionali storicamente sviluppati e studiati sono numerosi. Si va dal formalismo delle funzioni ricorsive, al lambda-calcolo, alle Unlimited Register Machines (URM), alle macchine di Turing. Le Macchine di Turing sono automi (nel senso descritto del testo di Ausiello, 2.5.1) ed in particolare automi con la massima potenza computazionale.

1.1 Nota sugli automi a stati finiti

La teoria degli automi è quella che viene utilizzata nella Teoria dei Linguaggi Formali per quanto riguarda l'approccio riconoscitivo alla definizione dei linguaggi. Le computazioni descritte dagli automi utilizzati nella teoria dei linguaggi formali sono computazioni che, preso un input, restituiscono un output di tipo YES o NO, oppure non terminano (si potrebbe far vedere che questo tipo di computazioni non sono concettualmente differenti da quelle

che, preso un input, restituiscono un output su un certo codominio). Nella teoria degli automi (come, del resto, in altri modelli computazionali) è possibile restringere il modello computazionale con dei vincoli che ne limitino la potenza espressiva dal punto di vista delle computazioni rappresentabili. Abbiamo per esempio, come indicato nel testo di Ausiello et al. 2.5.3 gli Automi a Stati Finiti, gli Automi a Pila, ecc.
Per quanto riguarda gli automi a stati finiti (vedi Ausiello et al. 3.1) che riconoscono le stringhe appartenti a linguaggi regolari, è possibile modificare leggermente questo formalismo in modo che le computazioni eseguite permettano, anzichè accettare o meno una stringa, di produrre una stringa su un opportuno insieme di caratteri di output. È sufficiente che, nella definizione di automa a stati finiti, si aggiunga un insieme Z di caratteri di output, si elimini la nozione di stato (o stati) finale e si faccia in modo che la funzione di transizione δ sia del tipo $\delta : \Sigma \times \mathrm{K} -> \mathrm{K} \times \mathrm{Z}$ Tale funzione permette ora di associare, al carattere correntemente letto ed allo stato interno attuale, lo stato successivo ed un carattere di output. Nella rappresentazione degli automi come diagramma degli stati (o grafo di transizione, chiamato anche diagramma di flusso in alcune soluzioni degli esercizi) basta quindi indicare sopra gli archi orientati non solo il carattere letto, ma anche il carattere di output prodotto dalla transizione rappresentata dall'arco orientato. Negli automi a stati finiti così descritti si può vedere che le computazioni effettuate possono prendere in input anche stringhe potenzialmente infinite, restituendo passo passo, un carattere per volta, stringhe di output infinite. Questa versione degli Automi a Stati Finiti è estremamente importante, poichè è con questi automi che si riesce a descrivere il comportamento dei circuiti sequenziali (che studierete nel corso di Architettura degli Elaboratori). Esistono inoltre procedure automatizzabili che, presa una specifica di funzionamento rappresentata attraverso un automa a stati finiti, permettono di sintetizzare il circuito sequenziale che realizza fisicamente tale specifica.

1.2 L'analisi di Alan Turing del concetto di *procedura effettiva*

Il tentativo di Turing di formalizzare **la classe di tutte le procedure effettive** risale al 1936 e portò alla nozione di Macchina di Turing. La sua importanza sta nel fatto che si tratta della prima anlisi che descriva come ha luogo la computazione; inoltre tale analisi portò ad una astrazione convincente ed ampiamente accettata del concetto di procedura effettiva. È degno di nota che questa analisi di Turing avvenne prima delle invenzioni dei moderni computer.

Diamo adesso un estratto della analisi di Turing. Si osservi che con il termine calcolatore Turing si riferisce ad un uomo che sta risolvendo un problema computazionale in maniera meccanica, e non ad una macchina.

> [...] Il procedimento del calcolo è normalmente realizzato scrivendo dei simboli su carta. [...] Supporò che la computazione avvenga su di un foglio unidimensionale, ovvero su di un nastro suddiviso in caselle.
> Supporò inoltre che il numero di simboli che possono essere scritti sia finito. [...] Le conseguenze di questa restrizione sul numero di simboli non sono rilevanti. È sempre possibile usare una sequenza di simboli al posto di un singolo simbolo. [...] Il comportamento del calcolatore in ogni istante è determinato dai simboli che egli sta osservando e dal suo stato mentale in quel dato istante. Possiamo supporre che ci sia un limite B al numero di simboli o celle che il calcolatore riesce ad osservare in un dato momento. Se vuole osservare qualcosa in più, dovrà effettuare più osservazioni successive. Supporremo anche che il numero di stati mentali di cui è necessario tenere conto sia finito. [...] Ancora una volta, tale restrizione non ha conseguenze rilevanti, poichè l'uso di stati mentali più complessi può essere evitato annotando più simboli sul nastro.
> Immaginiamo di scomporre le operazioni realizzate dal calcolatore in operazioni di base cosí elementari da risultare difficile una loro ulteriore scomposizione. Ciascuna di tali operazioni consisterà di un qualche cambiamento del sistema fisico costituito dal calcolatore e dal suo nastro. Conosciamo lo stato del sistema se conosciamo la sequenza dei simboli sul nastro, quali di essi sono osservati dal calcolatore, e lo stato mentale del calcolatore. Possiamo supporre che in ogni operazione di base non venga alterato più di un simbolo.
> [...] A parte questa possibilità di modificare simboli, le operazioni di base devono includere il cambiamento della distribuzione delle caselle osservate. [...] Penso sia raggionevole supporre che (le nuove celle) possano essere solo celle la cui distanza dalla più vicina delle celle precedentemente osservate non ecceda una certa quantità prefissata L.
> [...] Le operazioni di base devono perciò includere:
>
> 1. Cambiamento del simbolo di una delle caselle osservate.

2. Cambiamento di una delle caselle osservate con un'altra casella nello spazio di L celle di distanza da una di quelle precedentemente osservate.

Potrebbe succedere che qualcuno di questi cambiamenti comporti necessariamente il cambiamento dello stato mentale. Più in generale, le operazioni di base dovranno quindi essere le seguenti:

1. Un possibile cambiamento (a.) di un simbolo, insieme con un possibile cambiamento di stato mentale.
2. Un possibile cambiamento (b.) di una delle caselle osservate, insieme con un possibile cambiamento di stato mentale.

Quale operazione venga in effetti eseguita è determinato dallo stato mentale del calcolatore e dai simboli osservati.
[...] Possiamo adesso costruire una macchina che svolga il lavoro del calcolatore. Ad ogni stato mentale del calcolatore corrisponde una configurazione della macchina. La macchina osserverà le B celle corrispondenti alle B celle osservate dal calcolatore. Ad ogni mossa, la macchina potrà cambiare un simbolo su una delle celle osservate, o potrà spostare l'attenzione da una delle B celle osservate ad un'altra distante non più di L celle da una qualunque di quelle precedntemente osservate. [...]

Successivamente a questa analisi, Turing e Church indipententemente formularono la cosiddetta Tesi di Church-Turing:

> Ogni funzione intuitivamente computabile è computabile con una Macchina di Turing.

Tale tesi non si presta ad una dimostrazione matematica, poichè identifica una nozione intuitiva (quella di procedura effettiva), con un concetto matematico (quello di macchina di Turing). Tuttavia esistono varie ragioni per ritenerla valida, la più importante delle quali è il fatto che tutti gli altri modelli computazionali (di cui ne abbiamo elencati alcuni all'inizio della sezione) che sono stati proposti come formalizzazione del concetto di procedura effettiva, si sono rivelati essere equivalenti alle Macchine di Turing.
Nonostante le differenze nel formalismo, i vari modelli computazionali hanno delle caratteristiche comuni, quali:

- Esiste solo una quantità finita di "costrutti" che possono essere combinati per fornire una descrizione finita di ogni procedura effettiva.

- La computazione procede in maniera discreta, passo dopo passo.
- La computazione avviene in maniera deterministica, ovvero senza fare ricorso a metodi casuali (anche se esistono il realta' modelli computazionali non deterministici).
- Sebbene non vi sia un limite a priori sulla quantità di memoria o di tempo a disposizione, ogni computazione finita non deve basarsi su di una quantità infinita di tempo e/o spazio.
- Ogni passo nella computazione coinvolge solo una quantità finita di dati.

2 Le Macchine di Turing come modello computazionale

Uno dei primi modelli computazionali ad essere stato proposto è quello dovuto al matematico e logico inglese Alan Turing.
Nel 1936 Turing, nel tentativo di formalizzare la nozione di **procedura effettiva**, analizzò il comportamento di un essere umano quando risolve un problema di calcolo seguendo un metodo meccanico.
Egli riconobbe l'essenza di tale attività nella capacità di scrivere su un foglio (grande quanto necessario) usando un numero finito di simboli, con la possibilità di riconsiderare il lavoro già svolto. Per tenere traccia del lavoro già svolto, è opportuno non solo potere rileggere i simboli scritti in precedenza sul foglio, ma anche potere avere una sorta di *stato mentale*, influenzato da tutte le azioni precedenti e in grado di condizionare le scelte successive.
Da questa analisi Turing dedusse che una macchina per computare dovesse disporre:

- di un *nastro infinito*, suddiviso in celle, ciascuna capace di contenere un qualunque simbolo.
- di una *testina movibile*, atta a leggere e/o scrivere quintuple sul nastro.
- di una scatola nera, nota come *controllo finito*, per memorizzare lo stato mentale corrente.

Le operazioni elementari che una tale macchina doveva essere in grado di eseguire erano:

- cambiare il simbolo sotto la testina.

- spostare la testina in modo da agire su una delle celle adiacenti a quella corrente.

Secondo Turing era inoltre necessario che, congiuntamente a tali operazioni, la macchina fosse in grado di cambiare il proprio stato; pertanto assumeva che, più in generale, la macchina dovesse essere in grado, a seconda dello stato corrente e del simbolo letto dalla testina, di:

- cambiare il simbolo sotto la testina e lo stato corrente e spostare la testina di una cella.

Per consentire alla macchina di scegliere, in base alla situazione corrente, quale azione eseguire, era necessario dotarla di un **programma**, inteso come insieme di regole, dette **quintuple**, della forma (q1, a, b, M, q2) da interpretare come segue:

- Nello stato **q1**, leggendo il simbolo a, passa allo stato **q2** e scrivi il simbolo b sul nastro, poi sposta la testina a sinistra o a destra o rimani inerte, a seconda che **M** sia **S** (Sinistra), **D** (Destra) o **I** (Inerte).

Per descrivere la situazione in cui si trova una macchina di Turing in un certo momento basta fornirne la **Configurazione Istantanea**, ovvero il contenuto del nastro, la posizione della testina e lo stato corrente. Nell'ambito di questo modello una computazione altro non è che una sequenza di configurazioni istantanee, di cui la prima (detta **Configurazione Iniziale**) codifica l'input e lo stato iniziale, mentre ogni altra configurazione istantanea è ottenuta da quella immediatamente precedente eseguendo, di volta in volta, l'unica quintupla applicabile. Si noti che non necessariamente una tale sequenza è finita, poichè la macchina si arresta solo quando nessuna quintupla può essere applicata: in tal caso l'ultima configurazione (detta **Configurazione Finale**) è quella che codifica l'output.
È bene tener presente che esistono molte versioni (tutte computazionalmente equivalenti) della macchina di Turing. Per esempio si può definire la Macchina du Turing in modo che utilizzi quadruple anzichè quintuple. Analizzando le caratteristiche della Macchina di Turing, si nota che tale modello computazionale introduce, in maniera formale ma semplice e essenziale, alcuni elementi tipici della programmazione imperativa, quali:

locazione

Le celle del nastro di una Macchina di Turing hanno la stessa logica di funzionamento delle locazioni, in quanto contengono un *valore* che può essere aggiornato in maniera *distruttiva* (ovvero sostituito senza che sia possibile

risalire al valore precedente).

assegnamento

Le quintuple del tipo (q1, a, b, M, q2), che sostituiscono il simbolo **a** con il nuovo simbolo **b**, realizzano l'assegnamento del valore **b** alla locazione corrispondente alla cella sotto la testina.

memoria

Data la corrispondenza fra celle e locazioni, il nastro non è altro che una sorta di memoria lineare infinita ad accesso sequenziale, nel senso che per poter accedere ad una determinata locazione è necessario spostare la testina fino a raggiungere la cella corrispondente.

stato

Già nella definizione delle Macchine di Turing incontriamo il concetto di stato, come elemento che condiziona le scelte che la macchina opera durante il calcolo, e di configurazione, come descrizione della situazione in cui la macchina si trova in un dato istante.

iterazione

una computazione, dato il numero finito di quintuple avviene attraverso l'iterazione dell'*esecuzione* delle quintuple (che possono venire interpretate come una sorta di istruzioni della macchina).

3 La nascita del dualismo *Hardware-Software*

Nella concezione originale di Turing, i programmi per le Macchine di Turing (TM) erano parte integrante del controllo finito; per ogni problema che si volesse risolvere, bisognava definire un'opportuna TM, che poteva prendere in input i dati da elaborare, ma non poteva essere riprogrammata per eseguire un calcolo diverso da quello per cui era stata pensata. Ciò era in accordo con

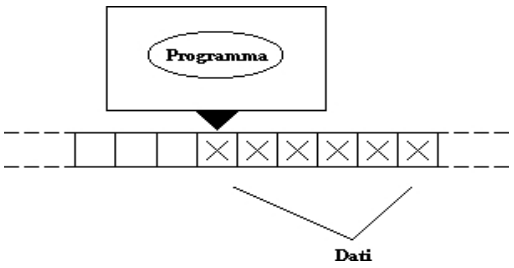

Figura 1: Una generica Macchina di Turing

l'intento iniziale di Turing, che non era quello di realizzare fisicamente le sue macchine, ma quello di determinare quale fosse l'insieme di funzioni per cui era possibile, in linea di principio, costruire un'apposita TM.
Successivamente Turing, considerando il fatto che una macchina di Turing si può rappresentare attraverso una stringa di simboli (basta dare una rappresentazione lineare della tabella che descrive la funzione di transizione, per esempio scrivendo una dopo l'altra le quintuple che corrispondono alle varie caselle della tabella di transizione), dimostrò che è possibile definire una Macchina di Turing Universale che, dati in ingresso la descrizione di una TM e il relativo input, simula il comportamento della prima: nasce così l'idea di una vera e propria macchina programmabile:

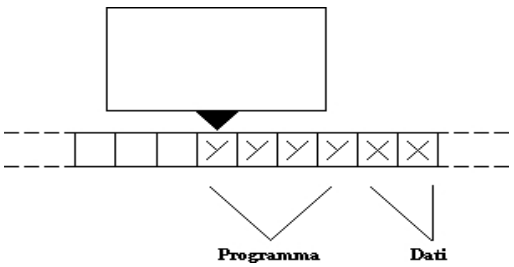

Figura 2: Macchina di Turing Universale

La dimostrazione dell'esistenza di una TM universale segna anche la nascita del dualismo *hardware-software*, cioè della contrapposizione fra ciò che è

fisicamente realizzato e ciò che è descrizione formale delle operazioni da svolgere. Da quanto detto sopra si nota come hardware e software nascono come equivalenti: quello che si può fare via software è anche realizzabile in hardware, e viceversa. La nozione di MT universale è concettualmente alla base del modello di architettura di Von Neuman (architettura **stored program**) che prende il nome da chi la propose, il matematico **John Von Neumann**. In tale tipo di architettura la CPU (Central Processing Unit) è preposta all'interpretazione del programma memorizzato in un'altra componente fisica, la RAM (*Random Access Memory*). Schematicamente:

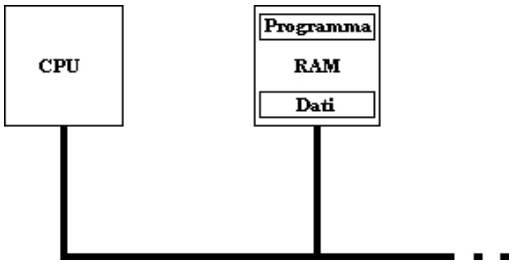

Figura 3: Architettura di Von Neumann

Già in questa architettura sono presenti *in nuce* tutte quelle componenti presenti nella maggior parte delle architetture sviluppate fino ai giorni nostri. È possibile fornire una descrizione astratta e, volendo, più dettagliata dei moduli che la compongono (per esempio analizzando le funzionalità della CPU si possono identificare ulteriori componenti al suo interno). Il nostro obiettivo nelle presenti note è proprio quello arrivare ad una definizione del concetto di Macchina Astratta Imperativa, una descrizione cioè il più generale possibile delle componenti che formano le architetture basate sul modello di Von Neuman, alla cui base abbiamo visto esserci il modello computazionale delle Macchine di Turing.
Si noti come non sia possibile fornire una definizione di Macchina Astratta in generale, poichè alla base di una Macchina Astratta c'è sempre un particolare Modello Computazionale. Le Macchine Astratte basate sul modello di Turing vengono dette imperative poichè alla base di tale modello c'è il concetto di comando (istruzione). Vedremo in seguito come esista una precisa corrispondenza tra Macchine Astratte e linguaggi di programmazione. I linguaggi di programmazione imperativi sono appunto quelli corrispondenti appunto a macchine astratte imperative.
Appare abbastanza evidente come, partendo da differenti modelli computazionali, si possa pervenire a differenti nozioni di Macchina Astratta (per esempio: funzionale, object-oriented, logica, parallelo/concorrente) L'attenzione che noi riserveremo alle macchine astratte imperative non è tanto do-

vuto alla "migliore qualita" del modello di Turing rispetto ad altri, ma al fatto che la stragrande maggioranza delle architetture concrete sono basate sul modello di Von Neuman e questo è dovuto a motivazioni puramente di carattere tecnologico.
Nella definizione di macchina di Turing, essendo questa un modello computazionale, tutto è ridotto al minimo necessario e descritto in modo molto formale. Se vogliamo arrivare ad una definizione di schema di macchina per calcolare basato su questo modello occorre isolare le "componenti" della macchina di Turing e descriverle in modo più generale possibile (attraverso un procedimento di "astrazione"). Quello che otteniamo non è più un modello computazionale, ma uno schema in cui poter inserire tutte le macchine basate sul modello computazionale imperativo (anche quelle che non calcolano tutte le funzioni calcolabili).
Per fare questo, non partiamo dalla definizione di macchina di Turing vera e propria, ma da quella di macchina di Turing Universale in cui abbiamo:

- una memoria (il nastro) che contiene dati e programmi. Questi sono rappresentati e strutturati in un modo semplicissimo nelle MT. Ma si può pensare di poterle strutturare e rappresentare in altro modo, magari utilizzando il supporto di particolari procedure e strutture, se ce ne fosse bisogno.

- un modo di recuperare gli argomenti delle operazioni. Nelle MT questo modo è semplicemente leggere quel che c'è sotto la testina. Però se vogliamo astrarre, possiamo pensare che, in generale, le operazioni possono recuperare gli argomenti in vari modi dalla memoria, utilizzando opportuni procedimenti e strutture.

- un modo per determinare qual'è la prossima istruzione da eseguire. In generale qualcosa che, servendosi magari di determinate strutture, determini qual è il flusso della sequenza di istruzioni da eseguire.

- qualcosa che, utilizzando tutto ciò che è a disposizione e coordinandolo opportunamente, permetta l'esecuzione dei programmi.

Sulla base di questo, nella prossima sezione daremo la nostra definizione di Macchina Astratta imperativa.

4 Il concetto di Macchina Astratta

Formalmente una **Macchina Astratta** (Imperativa[2]) è:

> *un insieme di strutture dati e algoritmi in grado di memorizzare ed eseguire programmi.*

Le componenti principali di una macchina astratta sono:

- un *interprete*
- una *memoria*, destinata a contenere il programma che deve essere eseguito e i dati su cui si sta operando.
- insieme di *operazioni primitive* (cioè funzionalità che si assume la macchina sia in grado di fornire) utili all'elaborazione dei dati primitivi (ovvero dati sui quali la macchina astratta sa lavorare).
- un insieme di operazioni e strutture dati che gestiscono il *flusso di controllo*, ovvero che governano l'ordine secondo il quale le operazioni, descritte dal programma, vengono eseguite.
- un insieme di operazioni e strutture dati per il *controllo del trasferimento dei dati*, che si occupa di recuperare gli operandi e memorizzare i risultai delle varie istruzioni.
- un insieme di operazioni e strutture dati per la *gestione della memoria*.

Le varie macchine astratte differiscono per il diverso modo di strutturare la memoria, per il diverso insieme di tipi primitivi che forniscono, per le diverse operazioni elementari che realizzano e per le diverse componenti di controllo e gestione; in generale cioè per il diverso modo di gestire l'esecuzione di un programma. La componente fondamentale che dá alla macchina astratta la capacità di eseguire programmi è l'**interprete**: esso coordina il lavoro delle altre parti della macchina astratta eseguendo un semplice ciclo *FETCH/EXECUTE*, finchè non viene eseguita una particolare istruzione primitiva (detta *HALT*) che ne provoca l'arresto immediato. Schematicamente: Il processo eseguito dall'interprete è suddiviso in fasi. Inizialmente avviene l'acquisizione, mediante l'uso delle strutture dati preposte al controllo della sequenza, della prossima istruzione da eseguire (*FETCH*). Quindi l'istruzione viene decodificata e si acquisiscono i suoi eventuali operandi, ricorrendo al

[2]D'ora in avanti ometteremo il termine "imperativo", visto che non ci occuperemo di macchine astratte di altro genere. Da sottolineare come la definizione fornita non sia assolutamente l'unica nè da considerarsi "la migliore".

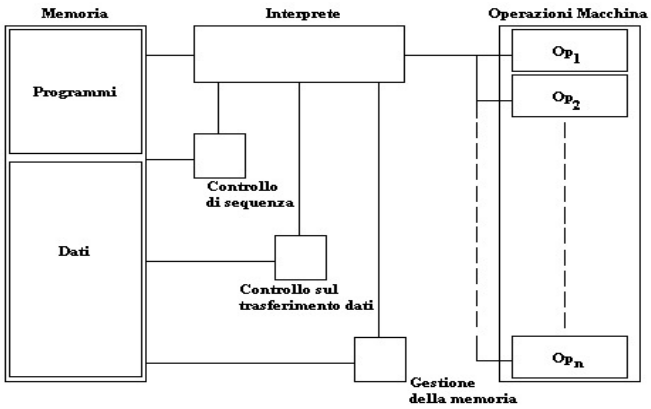

Figura 4: Struttura di una Macchina Astratta

controllo sul trasferimento dei dati. A questo punto viene eseguita l'opportuna operazione primitiva e l'eventuale risultato viene memorizzato (grazie di nuovo al controllo del trasferimento dei dati). Se l'operazione eseguita non è quella che fa arrestare l'interprete, il processo ricomincia. A titolo esemplificativo, vediamo alcuni esempi.

Macchina fisica con architettura tradizionale

Esaminiamo i componenti di una macchina astratta corrispondente ad una macchina fisica con architettura tradizionale (il vostro PC per esempio). Operazioni primitive tipiche di queste macchine sono le operazioni aritmetico-logiche, le operazioni di manipolazione di stringhe di bit, le operazioni di lettura e scrittura di celle di memoria e registri. Le strutture dati usate per il controllo sequenza sono il registro contatore istruzioni (PC) e le strutture contenenti i punti di ritorno dei sottoprogrammi. La sequenza di esecuzione è controllata da operazioni quali salti, condizionali, chiamate di e ritorni da sottoprogrammi. In una macchina hardware tradizionale, it controllo su trasferimento dei dati riguarda sostanzialmente le modalità di indirizzamento degli operandi di un'operazione e la memorizzazione del risultato relativo. A tale scopo vengono usati i registri indice e le operazioni che realizzano le modalità di indirizzamento indiretto.

Macchina fisica con architettura a pila

In una macchina con architettura a pila, la struttura dati fondamentale per il controllo sul trasferimento dei dati è una pila (stack), da cui sono prelevati gli operandi e su cui sono memorizzati i risultati. Nelle tradizionali macchine a

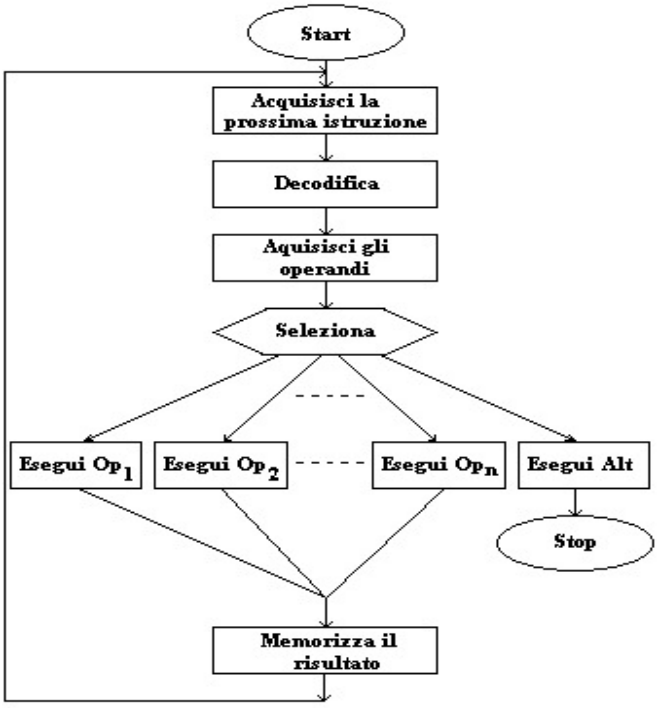

Figura 5: Il ciclo Fetch-Execute

registri, in cui i programmi ed i dati vengono memorizzati staticamente, non esiste praticamente gestione della memoria. Nelle macchine con architettura a pila questa consiste nella gestione della pila stessa.

Ristorante

La nozione di macchina astratta è ben più generica di quanto si possa credere. Infatti anche un ristorante, in un certo senso si può vedere come una macchina astratta. I programmi "eseguiti" da tale macchina sono composti da sequenze di ordinazioni di piatti (Es.: antipasto alla marinara; linguine al pesto; pepata di cozze; macedonia; limoncello). Supponiamo di avere un ristorante in cui ci sia un cuoco specializzato per ogni piatto ed un insieme di inservienti preposti a portare a tali cuochi gli ingredienti per i loro piatti. L'insieme dei cuochi può esser visto come l'inseme delle "operazioni" della macchina. La "memoria" della nostra macchina sarà il taccuino del cameriere. L' "interprete" del ristorante può essere il cameriere stesso che, letta la prima "istruzione" la "decodifica", dicendo agli inservienti quali ingredienti

portare a quale cuoco. Ovviamente anche la dispensa dovrà esser vista come parte della memoria del ristorante (la parte che memorizza gli "argomenti" delle istruzioni. Un altro cameriere provvederà a "memorizzare" sul nostro tavolo il risultato dell'esecuzione dell'istruzione. Ovviamente certi esempi di macchine astratte necessitano di un minimo di elasticità mentale per poter esser ricondotti al nostro schema formale. Teniamo presente inoltre che in realtà quella che abbiamo fornito è una descrizione della "realizzazione" della macchina astratta Ristorante.

Da notare come nella nostra definizione di macchina astratta non compaiano moduli per la gestione dell'Input/Output. Nulla ci vieterebbe di inserire anche questi nella nostra definizione. Essendoci però una enorme varietà di possibilità per l'Input/Output, tale concetto non si presta ad una generalizzazione ad un livello adeguato da poter essere inserito in una definizione che aspira ad essere la più generale possibile. Si potrebbe pensare di inserire operazioni particolari per l'I/O nell'insieme di operazioni fornite dalla macchina, oppure si potrebbe considerare una componente che gestisca l'interfacciamento della memoria con il "mondo esterno". Entrambe, e molte altre possibilità vanno bene. Fare una scelta particolare in una definizione generale come la nostra non troverebbe giustificazione rispetto alle altre possibilità.

5 Linguaggi di Programmazione e Macchine Astratte

Come visto nella definizione, una macchina astratta esegue programmi memorizzati al suo interno. Il *linguaggio macchina* (LM) di una macchina astratta M è il linguaggio in cui si esprimono tutti i programmi interpretabili dall'interprete di M. LM definisce quindi l'insieme delle strutture dati che realizzano la rappresentazione interna dei programmi eseguibili da M. È bene mettere in evidenza che quelli che normalmente chiamiamo programmi in LM non sono altro che particolari dati primitivi su cui opera l'interprete di M. Spesso però è più conveniente pensare ad un programma in termini di *stringhe di caratteri*. L'insieme di queste versioni dei programmi forma il cosiddetto $LMEST$ (*Linguaggio Macchina* ***EST****erno*).
È importante notare che sia il programma espresso in forma direttamente memorizzabile nella macchina, sia la sua versione mnemonica, sono due rappresentazioni dello stesso oggetto: **il programma astratto**. Quindi useremo il termine linguaggio macchina ad indicare indifferentemente LM o $LMEST$

(un esempio di *LM* sono i vari livelli di tensione nei bit di una memoria fisica, mentre il suo *LMEST* sarà un insieme di stringhe di caratteri "0" e "1"). Il compito di realizzare la conversione dal linguaggio *LMEST* al linguaggio *LM* è facilmente automatizzabile ed è svolto dal *LOADER* (così detto perchè carica il programma nella memoria della macchina astratta).
Così come data una macchina astratta resta definito un linguaggio di programmazione (il suo linguaggio macchina), analogamente dato un linguaggio di programmazione resta definita una macchina astratta che ne supporta le caratteristiche principali e che ha il dato linguaggio come suo linguaggio macchina. È da notare che gli *High Level Language* o *HLL* (cioè quei linguaggi che consentono al programmatore di esprimere i propri algoritmi in una forma molto vicina a quella in cui li ha pensati) definiscono macchine astratte tanto più complesse quanto maggiore è la potenza espressiva del linguaggio in questione. Ad esempio, un linguaggio che ha come tipo primitivo il tipo astratto *Lista*, avrà associata una macchina astratta che disporrà di opportune istruzioni macchina elementari per la realizzazione delle varie operazioni sulle liste (*Head*, *Tail*, *isNull*), e che avrà una organizzazione della memoria capace di gestire l'allocazione dinamica dello spazio per le liste in maniera automatica.
Di conseguenza la realizzazione in *hardware* è opportuna solo per macchina astratte relativamente semplici, mentre nel caso di una macchina astratta associata ad un linguaggio di alto livello questa scelta è poco conveniente, e risulta preferibile una realizzazione non hardware.
Nella prossima sezione affronteremo le varie possibilità di realizzazione di Macchine Astratte.

6 Realizzazione delle Macchine Astratte

Affinchè i programmi scritti in un dato linguaggio di programmazione L possano essere eseguiti, è necessario che la corrispondente macchina astratta venga implementata. Fondamentalmente è possibile adoperare tre tecniche:

- realizzazione hardware
- interpretazione (emulazione)
- traduzione (compilazione)

Realizzazione hardware

Con questa tecnica si realizzano in hardware tutte le componenti della macchina. In linea di principio tale tecnica è sempre possibile. E comprensibile

però che venga utilizzata solo macchina astratte relativamente semplici, a causa di considerazioni di carattere pratico, quali la scarsa flessibilità della realizzazione in hardware e l'elevato costo di progettazione e realizzazione in hardware di componenti dalle funzionalità molto complesse.
Ovviamente l'evoluzione delle tecniche di integrazione dei circuiti ha reso possibile (e renderà possibile sempre più in futuro) la realizzazione hardware di macchine sempre di maggior complessità.

Realizzazione interpretativa

A differenza della precedente, questa seconda tecnica, come anche la terza, richiede l'esistenza di una macchina già realizzata (*macchina ospite*). La tecnica interpretativa consiste nel realizzare tutte le componenti della macchina (cioè tutti gli algoritmi e le strutture dati che li definiscono) mediante programmi e strutture dati del linguaggio della macchina ospite. Di questo tipo di realizzazione ne esistono in realtà due tipi:

- emulazione via **firmware**
- emulazione via **software**

In linea di principio queste sono entrambe emulazioni software, la differenza consiste nel tipo della macchina ospite (microprogrammata nel primo caso) e nella realizzazione fisica della memoria che contiene i programmi che realizzano la le strutture dati e gli algoritmi della macchina. Nel primo caso la macchina ospite è realizzata in hardware e la parte programmi della memoria di tale macchina è una memoria ad alta velocità, solitamente di sola lettura, realizzata sullo stesso chip che contiene le altre componenti della macchina. Una tale macchina ospite è detta microprogrammata ed è estremamente semplice (le operazioni non sono altro che semplici operazioni su registri). I programmi di tale macchina ospite che realizzano gli algoritmi e le strutture dati della macchina che si vuole realizzare vengono chiamati *microprogrammi*. Nel secondo caso invece non c'è alcun vincolo su come sia realizzata la macchina ospite (e in particolare la sua memoria). *Le due tecniche di emulazione differiscono sostanzialmente nelle prestazioni della macchina astratta emulata*, in particolare rispetto alla velocità di esecuzione, che nel caso firmware è paragonabile a quella della realizzazione in hardware puro. Ovviamente la realizzazione nel caso firmware offre minore flessibilità in caso di future modifiche o estensioni della macchina realizata. È bene sottolineare che la realizzazione tramite firmware, molto in voga fino a non molti anni fa, sta lentamente tramontando, o si limita a porzioni sempre più piccole della macchina da realizzare. Questo a causa dell'affermarsi dei principi che sono alla

base delle architetture *RISC*. Quando parliamo di realizzazione interpretativa è bene fare sempre attenzione, quando parliamo di interprete, a quale macchina facciamo riferimento: la componente interprete della macchina che vogliamo ottenere è realizzata da un programma scritto nel linguaggio della macchina ospite la cui esecuzione è affidata alla componente interprete della macchina ospite.

Realizzazione compilativa

La tecnica compilativa si basa sulla idea di tradurre una volta per tutte l'intero programma scritto in L in un programma funzionalmente equivalente scritto nel linguaggio della macchina ospite. Il compito di eseguire tale traduzione è eseguito da un programma detto **compilatore**. In una realizzazione compilativa, uno si chiede: "si, ma dove sono le componenti della macchina che realizziamo?". In realtà è come se ci fossero, perchè noi dall'esterno vediamo esattamente gli effetti di una macchina realizzata.

7 Macchine Intermedie e Struttura a livelli dei computer moderni

La differenza di potenza espressiva fra una macchina astratta che vogliamo realizzare e la macchina che abbiamo a disposizione per tale realizzazione (*Macchina Ospite* o *HOST*) è detta **semantic gap**. Spesso il *semantic gap* tra la macchina da realizzare e la macchina *HOST* è talmente grande che è opportuno introdurre una o più macchine astratte intermedie.
Tipicamente, nell'implementare un linguaggio di programmazione (cioè la sua corrispondente macchina astratta) non si procede mai per pura compilazione o pura interpretazione su una data macchina *HOST*. La pura interpretazione potrebbe non essere soddisfacente a causa della scarsa efficienza della macchina realizzata emulando via software strutture dati, algoritmi e soprattutto l'interprete. Anche una realizzazione puramente compilativa a causa di un notevole *semantic gap* potrebbe portare a produrre programmi per la macchina ospite di dimensioni eccessive e magari lenti, senza considerare le difficoltà che si possono incontrare per sviluppare dei traduttori tra linguaggi eccessivamente diversi. Uno schema implementativo molto usato prevede fasi sia compilative che interpretative; la situazione, relativamente ad una realizzazione compilativa sopra una interpretativa è rappresentata nella figura seguente. Per colmare il divario fra la macchina *ML* e la macchina *HOST*, si progetta un'apposita *macchina intermedia* M_I che viene realizzata sulla macchina *HOST*. A questo punto la traduzione dei programmi in L avverrà

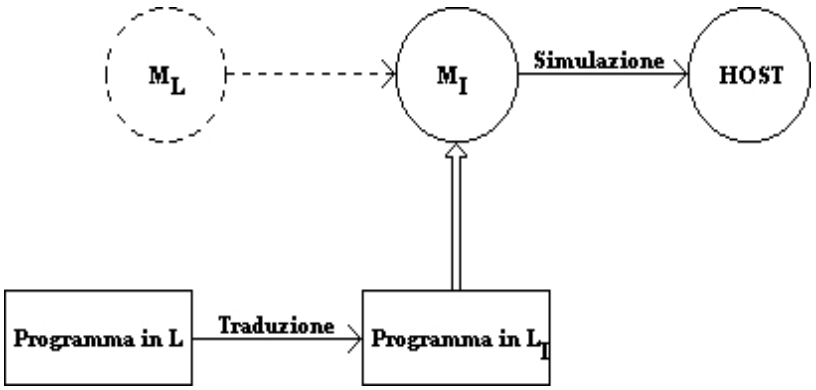

Figura 6: Implementazione di L mediante traduzione nel linguaggio di una macchina intermedia M_I e simulazione di M_I sulla macchina *HOST*.

in termini del linguaggio di $\mathbf{M_I}$, e successivamente il programma per M_I cosí ottenuto verrà eseguito tramite interpretazione sulla macchina *HOST*. Nel progetto della macchina intermedia bisogna tenere conto di due requisiti di efficienza:

1. velocità di simulazione della *Macchina Intermedia* M_I sulla macchina *HOST*.

2. compattezza del codice prodotto nel linguaggio della macchina M_I.

Per soddisfare tali requisiti, tipicamente M_I è solo un'estensione della macchina *HOST*, nel senso che ne condivide l'interprete, e ne potenzia alcuni aspetti mediante un insieme di routine note come *Run-Time System*, atto a colmare, negli aspetti, la differenza delle due macchine.
Tornando alla Fig. 6, notiamo che la pura interpretazione e la pura compilazione si hanno nei casi estremi caratterizzati rispettivamente dal fatto che la M_I coincida con la macchina ML o la macchina *HOST*.
Un esempio concreto di applicazione di questo schema si ha per il linguaggio di programmazione Java. In questo caso, la macchina intermedia M_I è la JVM (*Java Virtual Machine*[3]), il cui linguaggio è detto **Bytecode**, mentre il Run-Time System viene detto Java RunTime Enviroment. Per quanto riguarda la macchina *HOST* , nella pratica si hanno diverse realizzazione: dal PentiumII all'UltraSPARC, al PicoJavaII.
Non necessariamente la realizzazione di ML sulla macchina intermedia deve essere compilativa. Ci possono essere svariati motivi per avere sia ML che la

[3]È bene tener presente la possibile ambiguità che potrebbe generare questo nome. La Java Virtual Machine infatti non è la macchina astratta corrispondente al linguaggio JAVA, bensì la macchina intermedia per JAVA.

macchina intermedia realizzati entrambi per interpretazione.
Nello sviluppo dei nostri sistemi di calcolo in genere il procedimento descritto viene generalizzato, portando ad insieme di livelli di macchine astratte che si frappongono fra la macchina realizzata in hardware e la macchina astratta del "livello" utente, che potrebbe essere quella corrispondente ad un *HLL* o ad un particolare applicativo (per esempio il programma "Office" o il sistema di gestione utilizzato dalla vostra banca altro non sono che particolari macchine astratte).Gli scopi di tale stratificazione sono molteplici: gestire la complessità di progettazione, aumentare la flessibilità del sistema ecc.

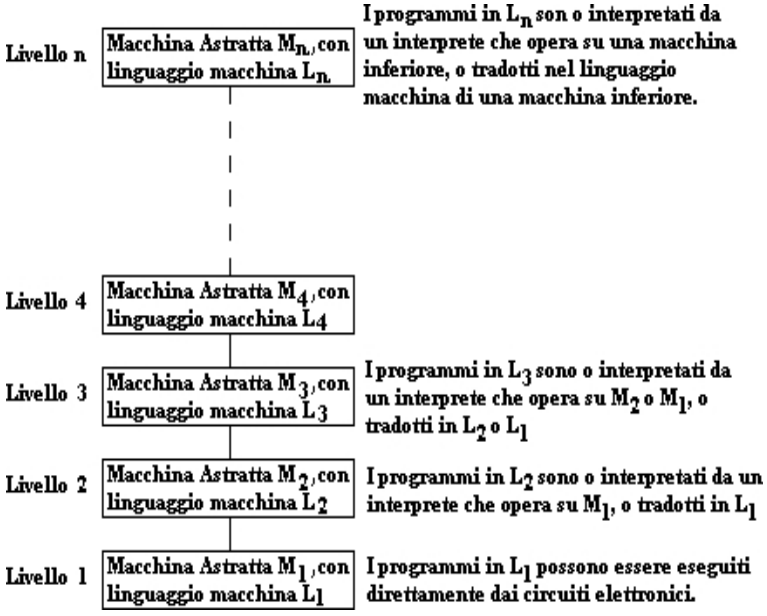

Figura 7: Gerarchia di macchine astratte

Il livello più basso rappresenta il computer reale e il linguaggio macchina che esso è in grado di eseguire direttamente. Ciascuno dei livelli superiori rappresenta una macchina astratta, i cui programmi devono essere o tradotti in termini di istruzioni di uno dei livelli inferiori (non necessariamente del livello immediatamente al di sotto), o interpretati da un programma che gira su di una macchina astratta di livello strettamente inferiore.
È da notare che nella pratica non è raro che un livello realizzi una macchina astratta che non copre del tutto il livello sottostante ma ne potenzia alcune delle caratteristiche, lasciandone trasparire del tutto altre.
È possibile, cioè, che alcune delle componenti della macchina siano esattamente quelle della macchina ospite, mentre altre siano completamente diverse o magari delle semplici estensioni. Nella seguente figura è schematizzata la

possibilità di una macchina, per esempio hardware, sulla quale è realizzata un'altra macchina attraverso emulazione via firmware solo di alcune sue componenti. Su quest'ultima è poi realizzata una ulteriore macchina che ha alcune componenti realizzate in software, mentre altre sono esattamente le stesse della macchina sottostante o di quella del livello ancora inferiore. Un semplice e chiaro esempio di livello che non copre i livelli sottostanti lo

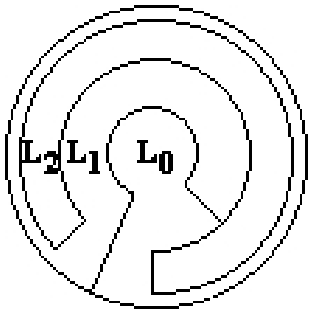

Figura 8: Il livello L_0 non del tutto coperto dai livelli superiori

potrete trovare quando affronterete il paragrafo 5.5.10 del Tanenbaum: The picoJava II Instructions. Vedrete come l'hardware dell'architettura picoJava II non realizza completamente le istruzioni del linguaggio JVM. Alcune di esse sono infatti ralizzate con microprogrammi, mentre altre, attraverso alune trap ad un software handler, sono realizzate in software. Quindi è come se esistesse una macchina astratta JVM_0 realizzata in hardware, sulla quale è realizzata una macchina JVM_1 e su quest'ultima la JVM completa. Ovviamente in tale semplice caso le parti firmware di JVM_1 e software si JVM non coprono alcuna parte di JVM_0, semplicemente la estendono. Un tipico computer moderno si può quindi pensare come una serie di macchine astratte realizzate una sopra l'altra, ciascuna in grado di fornire funzionalità via via più potenti. Per chiarire le idee sulla complessità di questa decomposizione in livelli, esaminiamo una possibile stratificazione dei comuni PC. È da notare tuttavia che non è in alcun modo necessario che un sistema di calcolo abbia un numero prefissato di livelli di astrazione: quella che segue è solo una possibile soluzione, che mostra come ogni livello colmi la sua parte di semantic gap. Nello schema non sono rappresentati i livelli sottostanti la **logica digitale**.
Le componenti fondamentali del *livello 0* sono le cosiddette porte logiche (*GATES*): a partire da raggruppamenti di esse si formano i registri e le memorie. Le *GATES* sono dispositivi elettronici che, pur appartenendo al mondo analogico, costituiscono il ponte verso il mondo digitale, poichè esse tipicamente realizzano funzionalità (Not, And, Or, ...) che sono ben forma-

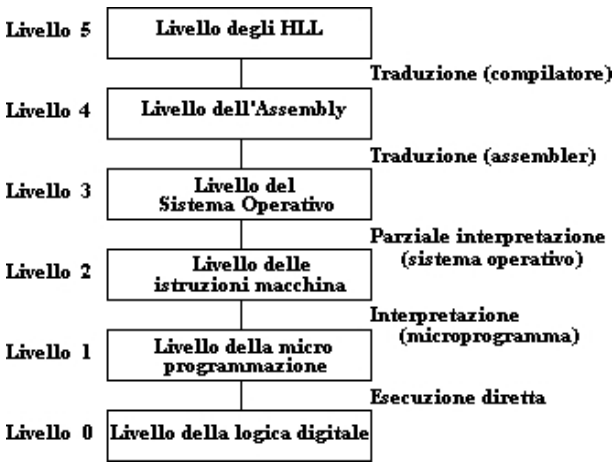

Figura 9: Un esempio di gerarchia di macchine astratte (di lato, il modo in cui sono realizzate)

lizzabili in termini di una teoria discreta nota come **Algebra di Boole**.
Collezioni di registri e circuiti digitali che realizzano funzionalità aritmetico-logiche sono alla base del livello 1. Qui troviamo altri elementi di interesse come il datapath e le strutture che presiedono al suo controllo. A questo livello inizia a diventare chiara l'idea di **flusso di informazione** poichè sequenze di bit viaggiano da una componente all'altra (della macchina astratta di questo livello) venendo eventualmente elaborate.
Il linguaggio del *livello 2* è quello cui tipicamente ci si riferisce quando si parla di **linguaggio macchina** di un certo computer. In termini tecnici, questo livello è noto come *ISA Level* (*Istruction Set Architecture Level*); tipiche istruzioni di questo livello sono *ADD*, *MOVE*, *SUB*, ...
Questo livello è di solito direttamente implementato in hardware nelle architetture *RISC*, mentre è tipicamente è realizzato mediante interpretazione firmware nelle architetture *CISC*.
Il *livello 3* evidenzia quella caratteristica precedentemente notata di non coprire del tutto i livelli sottostanti: si tratta del livello del **Sistema Operativo** (*SO*). In esso la maggior parte delle istruzioni dell'*ISA level* sono disponibili tali e quali; sono inoltre presenti nuove operazioni, che consentono un livello di astrazione notevolmente più alto nell'ambito della gestione della memoria e del flusso di controllo.
Tre delle principali funzionalità offerte dal Sistema Operativo sono:

- il concetto di *FILE*: a questo livello si realizza il salto qualitativo verso una memoria molto strutturata e organizzata gerarchicamente (**File System**) in cui informazioni complesse possono essere immagazzinate

e recuperate con grande facilità.

- la *MEMORIA VIRTUALE*: il Sistema Operativo offre ai livelli di astrazione superiore una macchina astratta con una quantità di memoria principale di gran lunga superiore a quella effettiva, liberando così il programmatore dai limiti fisici derivanti dalla particolare dotazione della macchina.

- l'astrazione di *PROCESSO* e il *MULTITASKING*: la possibilità di avere più "pezzi" di programma (**processi**) in esecuzione indipendente e (pseudo-)parallela è una delle caratteristiche che si è andata affermando negli anni; è tale meccanismo che rende possibili funzionalità quali la stampa in background o la garbage collection, etc.
  Questo in linea di principio significa che il livello del Sistema Operativo in realtà può mettere a disposizione non una singola, ma molteplici macchine astratte.

La programmazione dei livelli introdotti fino a questo momento è tipicamente compito di una cerchia ristretta di programmatori (i cosiddetti sistemisti), poichè si tratta di macchine astratte che sono state ideate non perchè interessanti in sè, ma in quanto necessarie per riuscire a gestire la complessità dell'implementazione delle macchine astratte di livello superiore.
Il *livello 4* segna quindi una rottura con i livelli precedenti: si tratta del livello dell'**assembly**, il primo che presenti qualche caratteristica (sebbene elementare) dei linguaggi di programmazione moderni (uso di etichette, mnemonici per le istruzioni, presenza di macro, variabili globali, procedure, etc...).
Tipicamente il livello 4 è realizzato sulle macchine sottostanti mediante il meccanismo di traduzione realizzato da un pezzo software (tipicamente scritto in ISA level language) noto come **ASSEMBLY language**. Tale livello si può definire come il più basso utilizzabile dal programmatore di applicazioni eseguite dal sistema di calcolo. Ad ogni modo è al *livello 5* che generalmente entrano in gioco gli HLL: C, Java, C++, Haskell, Prolog, etc... Si tratta del livello più vario, dove troviamo tutte le possibili soluzioni implementative, anche per macchine astratte relative allo stesso linguaggio: ad esempio, esistono varianti sia interpretate che compilate del BASIC.

Come accennato in precedenza l'avvento di Java ha portato alla nascita di un livello intermedio tra quelli del linguaggio ad alto livello e quello dell'assembly. Infatti prima di essere eseguiti, i programmi Java vengono tradotti in *bytecode*, ovvero nel linguaggio macchina della *Java Virtual Machine* (JVM), che è realizzata mediante interpretazione sui livelli sottostanti; successivamente saranno i bytecode così ottenuti che verranno eseguiti sulla *JVM*. I

vantaggi di questa tecnica hanno recentemente portato alla implementazione di molti linguaggi di programmazione (anche non Object Oriented) sulla JVM, piuttosto che sui livelli 2-3-4: sostanzialmente la JVM si sta affermando come macchina astratta standard all'interno della stratificazione tipica dei moderni sistemi di calcolo.
Teniamo a precisare ancora una volta che i livelli appena descritti sono una tra le infinite possibilità teoriche di strutturazione a livelli di un sistema di calcolo.

Programmazione delle Macchine Astratte

Riguardo la programmazione è talvolta possibile imbattersi (sempre più raramentente, per fortuna) in affermazioni del seguente tipo:

> Una programmazione consapevole ed adeguata per una particolare realizzazione di una macchina astratta è possibile solo se l'utente conosce non solo la struttura della macchina astratta, ma anche quali componenti sono stati realizzati direttamente in hardware, quali emulati, quali interpretati. Infatti, scegliere un modo piuttosto che un altro di realizzare un algoritmo può risolversi in un guadagno di prestazioni, dovuto alla maggiore efficienza della realizzazione di alcune componenti della macchina astratta rispetto ad altri.

Già il fatto che questa sia una affermazione eccessivamente generica (che non fa cioè riferimento a particolari contesti in cui poter essere utilizzata) dovrebbe insospettirci. L'Informatica è la metafora della vita e nella vita non si può fare alcuna affermazione che sia valida a prescindere da specifici contesti. Ergo, è possibile che in particolari contesti l'affermazione citata possa esser vera (pensiamo per esempio al caso di programmare una macchina astratta corrispondente al sistema di controllo della temperatura del nocciolo di una centrale atomica; ovviamente bisogna sfruttare al massimo tutte le possibilità di velocizzare la routine che, identificato un insolito aumento di temperatura, metta in moto tutte le procedure di sicurezza). In generale però sappiamo che non esiste solo la velocità di esecuzione di un programma come unico parametro di valutazione; per esempio c'è anche la trasportabilità di un programma. Un programma che abbia buone prestazioni solo perchè fa affidamento si certe caratteristiche della realizzazione della macchina astratta su cui gira, potrebbe avere prestazioni scadentissime su altre realizzazioni della stessa macchina. Questa considerazione è ancora più importante se si considera come sia sempre più rilevante oggi il concetto di codice mobile. In

generale quindi dobbiamo dire che nel programmare una macchina astratta dobbiamo far riferimento solo alla sua specifica, non alla sua realizzazione.

7.1 Al di sotto della Logica Digitale

Per poter inquadrare tutti gli aspetti dei sistemi di calcolo è possibile descrivere anche dei livelli sottostanti il livello 0. I livelli al di sotto della logica digitale costituiscono l'anello di congiunzione fra i fenomeni fisici e le tecniche per l'automazione del calcolo. Possiamo quindi identificare il **livello -1** (*Il livello dell'elettronica circuitale*) e quello **-2** (*Il livello della Fisica dello stato solido*). Pertanto, a *livello -2* i soggetti di interesse sono la struttura fisica dei solidi e le proprietà dei semiconduttori (o, in futuro, le proprietà di trasmissione della luce delle microfibre ottiche). Argomenti che appartengono prettamente all'ambito della Fisica e dei quali pertanto non ci si occupa in un corso di Architettura degli Elaboratori.
Sfruttando le conoscenze proprie del *livello -2* è possibile progettare e realizzare quei dispositivi elettronici (transistor, diodi, MOS, ...) che sono gli attori principali del *livello -1*. In questo livello si studia come combinare convenientemente più dispositivi elettronici per ottenere le cosiddette porte logiche (*GATES*), pervenendo così agli elementi di base del livello della logica digitale. È importante osservare che i *livelli -2* e *-1* sono dei livelli di astrazione fortemente diversi dai livelli al di sopra della logica digitale, poichè non si tratta di descrizione di Macchine Astratte. In altri termini, mentre è possibile parlare, ad esempio, della Macchina Astratta associata al *livello 2*, non ha senso parlare di Macchina Astratta del *livello -1* e *-2* (in realtà, come esercizio puramente speculativo, si può anche provare a vedere tali livelli come macchine astratte). Per rendere più chiara la differenza fra i meccanismi di astrazione al di sopra e al di sotto della logica digitale, possiamo considerare l'analogia seguente (che come tutte le analogie non informatiche è da prendere con le pinze). Vogliamo realizzare una **macchina astratta teatrale** a cui, dato in pasto il testo di un'opera teatrale (il programma) lo metta in scena (lo "esegue"). Per far questo prendiamo delle persone (il livello 0) e coordinandole attraverso l'uso di un regista otteniamo la nostra macchina (a livello 1). Utilizzando il livello 1 potremmo anche realizzare una macchina più complessa, una macchina cioè le cui istruzioni siano non le battute di un testo, ma titoli di intere opere. Questa "macchina" più complessa si può realizzare "estendendo quella del livello 1" attraverso una biblioteca di opere e qualcuno che preso il titolo di un'opera lo "interpreti" andando nello scaffale giusto a prendere il testo dell'opera in questione e portando il testo al regista. Ora in questo "sistema a livelli", se volessimo considerare dei livelli inferiori allo 0 quali potrebbero essere? Essendo delle persone gli elementi del

livello 0, il ***livello -1*** *potrebbe essere quello degli organi che compongono un persona umana,* ***quello -2*** *quello dei tessuti che compongono i vari organi,* ***quello -3*** *quello delle cellule.*
Analizzando la gerarchia appena descritta nella sua totalità, appare evidente come, sebbene si tratti di livelli di astrazione costruiti intorno all'uomo, il meccanismo di astrazione su cui si basano i livelli è del tutto diverso al di sopra e al di sotto di esso.
Ritornando alla stratificazione dei sistemi di calcolo, ci si rende conto che i livelli -1 e -2 sono utili perchè ci aiutano a capire il funzionamento del livello della logica digitale, ma non perchè costituiscano esempi di realizzazione di Macchime Astratte, in quanto sono basati su un meccanismo di astrazione differente. è anche per questo motivo, oltre che per quelli già citati precedentemente, che non ce ne occupiamo in dettaglio.

8 Programmazione e Livelli di Astrazione

La gerarchia di macchine astratte di un sistema di calcolo non termina necessariamente con l'implementazione di un linguaggio di programmazione *HLL* L. Infatti un qualunque programma in L aggiunge un ulteriore livello alla gerarchia estendendo la macchina astratta associata a L (o nel caso L non sia interpretato, *la macchina intermedia di* L) in alcuni dei suoi componenti, simulando generalmente nuove operazioni e nuovi tipi di dato. Da questo punto di vista quindi qualunque attività di programmazione può essere considerata come una attività di definizione e di realizzazione di macchine astratte.
Vedere la programmazione in un certo linguaggio come estensione della macchina astratta ad esso associata evidenzia il fatto che la potenza e la flessibilità di un linguaggio di programmazione dipendono dai meccanismi di astrazione che il linguaggio stesso offre. Praticamente tutti i linguaggi forniscono un meccanismo di astrazione (sia esso procedura, funzione o subroutine), che permette di definire nuove operazioni. Solo alcuni linguaggi (tipicamente quelli **object-oriented**) possiedono meccanismi soddisfacenti per la definizione di nuovi tipi di dato astratti. Infine la possibilità di estendere le altre componenti della macchina astratta (*contollo di sequenza*, *controllo sul trasferimento dati*, *gestione della memoria*), non è stata ancora realizzata in nessuno dei linguaggi di programmazione maggiormente in uso. Si potrebbe pensare che, anche se il linguaggio non fornisce un particolare meccanismo di astrazione, sia comunque possibile simulare in esso un qualunque costrutto; tuttavia in questo caso l'estensione risulterà di più complessa realizzazione per il programmatore e di più difficile utilizzo da parte dell'utente rispetto

alla corrispondente estensione ottenuta con un linguaggio di programmazione che possiede meccanismi di astrazione adeguati.