L'ARCHITETTURA DI VON NEUMANN

Dispense del corso di Fondamenti di Informatica I

Alberto Marchetti-Spaccamela

Indice

1 Cosa è un computer: un po' di terminologia **1**
1.1 Hardware 1
1.2 Codici per la rappresentazione delle informazioni 2
1.3 I linguaggi di programmazione e il software 3
1.4 Categorie di computer 4

2 L'architettura di von Neumann **5**
2.1 Il sottosistema di memoria 6
2.1.1 Operazioni di memoria 6
2.2 L'unità centrale di elaborazione e il ciclo di esecuzione di un'istruzione 8
2.2.1 La fase di reperimento dell'istruzione (Fetch) 9
2.2.2 La fase di decodifica 10
2.2.3 La fase di esecuzione e l'unità aritmetico/logica (ALU) 10

3 Un semplice linguaggio macchina **11**

4 La legge di Moore **17**
4.1 I limiti estremi della legge di Moore 21
4.2 Conclusioni 23

1 Cosa è un computer: un po' di terminologia

Un computer (in italiano elaboratore o calcolatore elettronico) è un dispositivo elettronico, che comunica attraverso dispositivi di ingresso e di uscita o tramite reti di comunicazione e che esegue programmi immagazzinati nella memoria.

Un **programma** é una sequenza di istruzioni che puó ricevere in ingresso dei dati, forniti tramite dispositivi di ingresso (input) o presenti in memoria e produce in uscita risultati che possono essere dati a dispositivi di uscita (output) o possono essere memorizzati per uso futuro su dispositivi di memoria. I programmi sono scritti in un linguaggio di programmazione le cui istruzioni sono o direttamente eseguibili dal computer o sono tradotte in istruzioni prima dell'esecuzione. Linguaggi di programmazione comprendono ad esempio Python, C, C++, Java.

I dati in ingresso e in uscita sono forniti secondo diversi formati (testo, numeri, immagini, audio, video ecc.) e sono elaborati e trasformati in informazioni utili per l'utente. Per questa ragione spesso, con il termine elaborazione delle informazioni, si fa riferimento alla successione di input, elaborazione e memorizzazione dei risultati.

I computer sono stati ideati inizialmente per eseguire in modo automatico calcoli di tipo scientifico (o per scopo militare). Successivamente l'evoluzione maggiore dei computer si é avuta in ambito civile. Un elemento importante di questo sviluppo é la possibilitá di comunicazione: la maggior parte dei computer di oggi comunicano con altri computer. Come risultato, le comunicazioni sono diventate un elemento essenziale della ciclo di elaborazione delle informazioni. Per questa ragione il termine italiano calcolatore elettronico per denotare un computer appare oggi non adeguato: il termine elaboratore elettronico meglio esprime l'elemento essenziale dei computer come dispostivi in grado di elaborare informazioni.

1.1 Hardware

Dal punto di vista fisico un computer è un oggetto molto sofisticato composto da dispositivi elettrici, elettronici, e componenti meccanici che rappresentano la parte **hardware**. Questi componenti includono i dispositivi di input e di output, una unità di elaborazione centrale, dispositivi di memoria e di comunicazione.

Un **dispositivo di input** è un qualsiasi componente hardware che consente di immettere i dati e le istruzioni in un computer. Alcuni esempi di dispositivi di input molto comuni sono la tastiera, il mouse, il microfono, lo scanner, la web cam.

Un **dispositivo di output** è una componente hardware che fornisce le informazioni risultato dell'elaborazione ad una o più persone. Tra i dispositivi di uscita comunemente usati ricordiamo la stampante, il video, gli altoparlanti.

L'unità centrale di elaborazione contiene i componenti elettronici del computer che vengono utilizzati per l'elaborazione. I circuiti e i dispositivi elettronici dell'unità centrale sono dei circuiti integrati elettronici di solito posti su un circuito chiamato **scheda madre**. Due componenti fondamentali presenti sulla scheda madre sono il **processore e la memoria**. Il processore, detto anche **CPU (Central Processing Uniti - unità di elaborazione centrale)**, è il componente elettronico che interpreta ed esegue le istruzioni di base con cui si elaborano le informazioni e che sono dette **istruzioni macchina**. La CPU è un circuito elettronico detto **chip** delle dimensioni di pochi centimetri.

La **memoria** è costituita da componenti elettroniche che memorizzano le istruzioni e i dati necessari per tali istruzioni. La memoria può essere permanente, o volatile; una memoria

volatile perde le informazioni in essa contenute quando il computer è spento. La memoria presente sulla scheda madre è anche detta **memoria principale** per distinguerla dalla **memoria secondaria**. come, ad esempio, le penne USB, i dischi rigidi, i dischi ottici e le schede di memoria flash. I diversi tipi di memoria si caratterizzano per tre principali caratteristiche: la capacità di memoria, la velocità di lettura e scrittura e il costo per unità di informazione.

Un **dispositivo di comunicazione** consente al computer di trasmettere e ricevere dati, istruzioni e informazioni da e verso uno o più computer o dispositivi mobili. Le comunicazioni avvengono tramite cavi, linee telefoniche, reti cellulari, satelliti e altri mezzi di trasmissione. Alcuni mezzi di trasmissione media, come i satelliti e le reti radio cellulari, sono senza fili (*wireless*).

Una **rete** è un insieme di computer e dispositivi collegati tra loro tramite dispositivi di comunicazione; le reti consentono ai computer di condividere le risorse, come hardware, software, dati e informazioni. In molte reti, uno o più computer, detti *server* hanno un ruolo particolare. Ad esempio in una rete il server può controllare l'accesso alle risorse su una rete agli altri computer connessi in rete, e che sono detti *client*. Oppure il server può memorizzare archivi centrali di dati (*database*) e fornire risposta ad interrogazioni da parte dei client.

Internet è un insieme di reti che collega imprese, governi enti, istituzioni e individui; più di un miliardo di persone in tutto il mondo usa giornalmente Internet per una varietà di motivi di lavoro, di informazione e per attività sociali.

1.2 Codici per la rappresentazione delle informazioni

Il **bit** è la più piccola unità di memorizzazione e può assumere uno di due valori: 1 e 0.

Un **byte** indica una sequenza di otto bit; quattro byte (32 bit) sono una **parola**; nessuno ha ancora inventato un termine accettato da tutti per indicare una sequenza di 64 bit.

La memoria è organizzata in una sequenza di **celle**, ognuna delle quali può contenere una sequenza di bit; ogni cella è identificata da un indirizzo. Il numero di bit di una cella di memoria dipende dal processore; tipicamente le architetture di computer presenti attualmente sul mercato utilizzano celle di 32 o 64 bit.

Poiché l'elaborazione delle informazioni richiede la memorizzazione e il trattamento di dati complessi (numeri, informazioni testuali, suoni, immagini ecc.) è necessario trovare metodi opportuni per rappresentare informazioni diverse. È importante ricordare che una cella di memoria rappresenta solo una sequenza di bit; si pone quindi il problema di dare un significato, cioè di come assegnare un contenuto informativo utile ad una sequenza di 0 e 1.

Ad esempio supponiamo che il telefono di Mario sia 333 2221100; la memorizzazione del numero di Mario nella nostra agenda telefonica è una sequenza di bit. In altre parole dobbiamo essere in grado di interpretare il contenuto di una o più celle di memoria come *"Nome: Mario, Telefono 3332221100"*.

In particolare, le informazioni complesse e le istruzioni del processore sono rappresentati in memoria e negli altri dispositivi di un elaboratore utilizzando opportuni **codici**; un codice associa ad ogni simbolo di un interesse una sequenza di bit. Ad esempio, il **codice ASCII** permette di rappresentare informazioni di tipo testo: ad ogni carattere dell'alfabeto il codice ASCII associa un byte (cioè una sequenza di otto bit)[1].

[1]Esistono diverse varianti del codice ASCII per poter rappresentare con soli otto bit i caratteri più usati nelle varie lingue (includendo ad esempio i caratteri cirillici, i caratteri dell'alfabeto greco, ebraico, e di molte lingue orientali. Data l'importanza dell'alfabeto latino i caratteri latini sono sempre presenti e codificati stesso modo; ad esempio il carattere "a" è sempre rappresentato con la sequenza 0110001.

Ciascun codice specifica le regole che ci permettono di interpretare una sequenza di zeri e uni e dare ad essa un significato utile. Poiché le informazioni che utilizziamo sono di molti tipi abbiamo bisogno di codici diversi. Ad esempio, per rappresentare una immagine sullo schermo del nostro telefonino, si utilizza un codice che permette di interpretare una sequenza di bit come una immagine a colori; analogamente quando ascoltiamo un brano musicale memorizzato su un personal computer (o un CD) si utilizza un codice che interpreta sequenze di bit come suoni[2].

1.3 I linguaggi di programmazione e il software

I programmi che sono eseguiti dai computer sono anche detti **software**, e consistono di una sequenza di istruzioni correlate, organizzate per un scopo comune, e scritte in un linguaggio che specifica le operazioni da eseguire sui dati.

Il processore è in grado di eseguire direttamente programmi composti di istruzioni che realizzano operazioni molto semplici: l'insieme di tali istruzioni è detto **linguaggio macchina**.

Le istruzioni del linguaggio macchina sono elementari e per questa ragione scrivere programmi nel linguaggio macchina è molto complicato perché richiede di scomporre i singoli passi dell'algoritmo di soluzione in una sequenza di operazioni semplici vicine alla logica della macchina piuttosto che alla logica con cui siamo abituati a ragionare. Inoltre poiché le istruzioni del linguaggio macchina sono elementari ne consegue che la realizzazione di programmi complessi in grado di eseguire operazioni sofisticate sui dati richiede la scrittura di programmi composti da molte istruzioni (in alcuni casi anche milioni di istruzioni).

Un altro limite di scrivere programmi in linguaggio macchina è dato dal fatto che ogni macchina ha il suo linguaggio e, quindi, cambiare l'elaboratore potrebbe richiedere di dover riscrivere completamente i programmi.

Le diffcoltà precedenti rendono praticamente impossibile riuscire a scrivere programmi molto complessi (come, ad esempio, un browser web o un'applicazione come Skype) in linguaggio macchina. La soluzione che è utilizzata oggi è stata proposta negli anni cinquanta del secolo scorso (quando i ricercatori hanno iniziato ad utilizzare i primi computer realizzati). L'idea è di progettare linguaggi di programmazione con istruzioni più potenti e che permettono di scrivere programmi in meno tempo e con minori possibilità di errori.

Linguaggi di questo tipo sono i **linguaggi ad alto livello**. Esempi di linguaggi ad alto livello sono i linguaggi C, C++, Java, Python. Le istruzioni dei linguaggi ad alto livello non possono essere direttamente eseguite dal processore ma devono essere tradotte in una equivalente sequenza di istruzioni del linguaggio macchina. Ad esempio una istruzione del linguaggio Python permette di ordinare una sequenza di numeri; questa istruzione equivale ad una sequenza di decine di istruzioni nel linguaggio macchina.

Per effettuare la traduzione dei programmi scritti linguaggio ad alto livello in programmi in linguaggio macchina sono utilizzati programmi traduttori che possono essere di due tipi **compilatori** e **interpreti**. I compilatori sono programmi che effettuano la traduzione del programma scritto dal programmatore in linguaggio macchina. Gli interpreti invece procedono alla traduzione dei programmi istruzione per istruzione: ogni nuova istruzione del programma viene interpretata ed eseguita.

[2] È noto che esistono diversi codici per immagini e per il suono a seconda della qualità desiderata e che maggiore è la qualità maggiore è anche la dimensione dei dati. Ad esempio, il codice MP3 permette di memorizzare file musicali con qualità inferiore al codice utilizzato sui CD ma richiede molta meno memoria.

Il **software di sistema** è costituito dai programmi per il controllo e la gestione del computer e delle sue periferiche. In particolare il **sistema operativo** è un insieme di programmi che coordina tutte le attività tra i dispositivi hardware del computer e permette inoltre di comunicare con il computer e altri software (due esempi di sistema operativo sono Windows per i computer e Android per i cellulari). Un programma di utilità consente all'utente di eseguire le attività di manutenzione di tipo normalmente relativi alla gestione di un computer, sui suoi dispositivi, o i suoi programmi. La maggior parte dei sistemi operativi includono diversi programmi di utilità per la gestione del dispositivi di ingresso/uscita e per le comunicazioni.

Il **software applicativo** è costituito da programmi progettati per rendere gli utenti più produttivi nel lavoro o per realizzare attività personali come, ad esempio, il browser Web, che consente utenti con una connessione a Internet per accedere e visualizzare le pagine Web.

1.4 Categorie di computer

Gli esperti del settore tipicamente classificano i computer in diverse categorie: personal computer (desktop), computer portatili e dispositivi mobili, giochi console, server, mainframe, supercomputer, ed embedded computer. Non ci interessa in questa sede stabilire i criteri di classificazione dei diversi tipi di computer anche perché, a causa della rapida evoluzione della tecnologia, la distinzione tra le categorie non è sempre chiara. Ci limitiamo ad osservare che i computer presenti sui diversi dispositivi (ad esempio un telefonino, un videogioco o un personal computer) sono molto simili tra loro e si differenziano essenzialmente per dimensioni, velocità, potenza di elaborazione, e prezzo ma sono molto simili per quanto riguarda le tecnologie e i criteri di progettazione e i linguaggi di programmazione utilizzabili.

Concludiamo questa sezione osservando che lo sviluppo tecnologico degli ultmi venti anni ha reso disponibili unità centrale di elaborazione di ridotte capacità ad un prezzo molto basso (pochi euro). Questo fenomeno ha permesso una pervasività dei computer fino a qualche anno fa non immaginabile.

In particolare un **computer embedded** è un particolare computer configurato come un componente in un prodotto. Computer embedded sono ovunque nelle case, nelle automobili e al lavoro. Ad esempio, un'automobile venduta oggi contiene diversi computer embedded che sono responsabili della sicurezza e del corretto funzionamento dell'auto. I computer embedded sono dispositivi di potenza computazionale limitata con scopi precisi specificati dalla particolare applicazione per cui sono stati progettati. Il seguente elenco identifica una varietà di prodotti che contengono computer embedded:

- Elettronica di consumo: telefoni cellulari, televisori digitali, macchine fotografiche, videoregistratori, lettori DVD e registratori, segreterie telefoniche
- Dispositivi di automazione: termostati, ascensori, dispositivi di monitoraggio della sicurezza di impianti, apparecchi, luci
- Automobili: freni antibloccaggio (ABS), controllo del motore, dell'airbag
- Controllori di processo e robotica: sistemi di monitoraggio remoti, controllo della potenza, controllo di macchine, dispositivi medici
- Dispositivi per computer e macchine per ufficio: stampanti, fax e fotocopiatrici

Per dare un'idea delle dimensioni e dell'impatto economico dei sistemi embedded osserviamo che nel 2010 si stima che siano stati venduti circa 400 milioni di personal computer nel mondo.

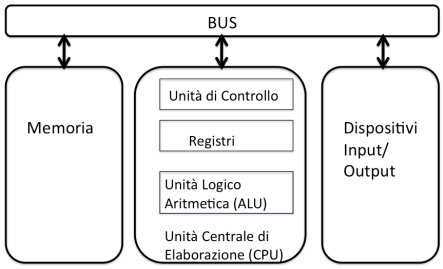

Figure 1: L'architettura di von Neumann

I sistemi embedded venduti nello stesso anno sono stati invece circa 16 miliardi (quindi circa quaranta calcolatori embedded per ciascun personal computer). Si prevede inoltre che - mentre il mercato dei personal computer è un mercato cosidetto maturo che non dovrebbe avere significativi incrementi nei prossimi anni - per il mercato dei sistemi embedded si prevede una crescita di circa il 14 % annuale. Se questa previsione si avvera nel 2020 saranno venduti 100 calcolatori embedded per ogni computer ad uso personale.

2 L'architettura di von Neumann

L'architettura di von Neumann prende nome dal matematico e informatico John von Neumann il quale inizialmente ha contribuito a svilupparla per l'EDVAC, uno dei primi computer elettronici costruiti. John von Neumann non è stato il principale progettista dell'EDVAC, ma quello che ha descritto la sua architettura in una relazione scritta nel 1945. In questo modo l'articolo di von Neumann è diventato il testo di riferimento per una nuova generazione di computer basati sull'architettura hardware dell'EDVAC. Di conseguenza tale architettura è diventata nota come "architettura di von Neumann" suscitando malcontento tra gli altri progettisti dell'EDVAC. Osserviamo anche che non è stato l'EDVAC il primo computer operativo della storia basato sull'architettura di von Neumann[3].

Praticamente tutti i moderni sistemi informatici, al più alto livello di astrazione, sono conformi alla architettura di von Neumann dal punto di vista logico; questo fatto è sorprendente se pensiamo agli sviluppi tecnologici avvenuti nel frattempo. La figura 1 illustra la struttura logica di un sistema informatico moderno.

Le componenti sono:

- **L'unitá di controllo (UC)**, che, come suggerisce il nome, ha la responsabilitá generale del sistema.
- **L'unitá aritmetico-logica (Arithmetic Logic Unit, ALU)** che, come il nome suggerisce, esegue le operazioni aritmetiche (addizione, sottrazione, ecc) e logiche (AND, OR, ecc).
- **I registri (REG)** che formano una piccola memoria all'interno della CPU. Questa é l'unica memoria a cui la ALU ha accesso diretto. Per eseguire le operazioni i dati devono essere trasferiti nei registri.

[3]Esiste una corrispondenza scritta per cui l'idea originaria dovrebbe essere accreditata a Alan Turing.

- **La Memoria** dove risiede sia il codice del programma che i dati, durante l'esecuzione del programma.
- **Dispositivi di ingresso e di uscita** (indicati anche come dispositvi I/O) sono il sistema di comunicazione con il mondo esterno esterno, in questo caso, a significare al di fuori del sottosistema CPU-Memoria. Sul computer di oggi alcuni operazioni di I/O avvengono all'interno del sistema nel suo complesso (ad esempio letture e scritture su dischi, CD-ROM), mentre alcune operazioni si svolgono tra il sistema e il mondo esterno, ad esempio attraverso una rete.

Le componenti precedenti comunicano fra loro tramite un **bus**[4]. Le architetture moderne prevedono più bus; in questo modo si raggiunge una maggiore velocità di comunicazione e, quindi, una maggiore velocità di calcolo.

Dispositivi I/O sono oltre lo scopo di questa breve introduzione e, per questa ragione, non affronteremo in dettaglio gli aspetti relativi alla loro realizzazione. Nel seguito focalizzeremo la nostra attenzione sulla memoria e la CPU.

2.1 Il sottosistema di memoria

La memoria dove il computer memorizza le informazioni e (in esecuzione) programmi e si distingue in **memoria principale** e **memoria secondaria**, detta anche **memoria di massa**. Ricordiamo che le informazioni memorizzate sono sequenze di bit.

La memoria principale è il deposito primario dei dati e delle istruzioni dei programmi in esecuzione ed è di due tipi diversi: la **RAM (Random Access Memory)** e la **ROM (Read Only Memory)**. La RAM è un tipo di memoria dove effettuare operazioni di lettura del contenuto della cella di memoria e di scrittura con cui modifichiamo il contenuto. La ROM è una memoria di sola lettura. Un'altra differenza fra i due tipi è che la ROM é una memoria permanente, mentre la RAM è volatile (cioè le informazioni sono perse quando la corrente elettrica è interrotta).

La memoria secondaria, con cui ci si riferisce e diversi dispositivi quali le unità disco, i dispositivi USB e simili, possono contenere grandi quantità di dati. Le maggiori differenze tra memoria principale e secondaria sono il tempo di accesso e il costo e la permanenza dei dati: la memoria RAM è più veloce e più costosa e, come abbiamo visto, volatile.

2.1.1 Operazioni di memoria

Ci sono due importanti attributi associati a ogni cella di memoria:

- l'indirizzo di memoria
- il contenuto

L'indirizzo di memoria è analogo all'indirizzo di casa: ci dice come trovare la cella di memoria, ma non ciò che contiene. Ad esempio, l'indirizzo "via Ariosto 25, Roma" ci permette di trovare immediatamente l'edificio ma l'indirizzo non fornisce informazioni su chi troveremo; per sapere chi vive o lavora a via Ariosto 25 dobbiamo andare all'interno dell'edificio. Possiamo affermare che il contenuto della cella è l'equivalente delle persone che sono nell'edificio.

[4]Osserviamo che la descrizione originale dell'architettura di von Neumann non prevedeva un bus ma collegamenti diretti fra le diverse componenti. L'introduzione del bus è avvenuta successivamente

In altre parole si deve guardare nella cella di memoria per conoscere il contenuto: non c'è modo di conoscere il contenuto della cella dal suo indirizzo (e, viceversa, conoscere l'indirizzo dal contenuto della cella).

Si può dedurre dalla discussione precedente che la memoria deve essere in grado di fare due cose:

1. Fornire il contenuto corrente di una cella di memoria specificata dalla CPU
2. Memorizzare nuovi contenuti in una specificata cella di memoria.

Per la prima operazione (lettura) la CPU deve passare due informazioni al sottosistema di memoria:

1. L'indirizzo della cella di memoria da leggere.
2. L'indicazione che l'operazione richiesta sia una lettura.

Il sottosistema di memoria, dopo aver recuperato l'informazione richiesta, dovrà poi inoltrarla alla CPU.

Al fine di effettuare la seconda operazione (scrittura), la CPU deve passare:

1. L'indirizzo della cella di memoria da leggere
2. I contenuti da memorizzare
3. Una indicazione che viene richiesta una scrittura.

Per eseguire le due operazioni di lettura e scrittura, si utilizza l'interfaccia fra la CPU e la memoria descritta in figura 2 e composta da:

1. Il **Registro Indirizzo Memoria (Memory Address Register, MAR)**, che contiene l'indirizzo della cella richiesta, sia per l'operazione di lettura o scrittura.
2. Il **Registro Dati della Memoria (Memory Data Register, MDR)** che viene utilizzato per lo scambio di informazioni dei dati; in particolare il sottosistema di memoria colloca il contenuto recuperato dalla memoria (lettura) e la CPU pone il nuovo contenuto della memoria (scrittura).
3. Due linee in cui passano i comandi con cui l'unità di controllo specifica l'operazione richiesta (una linea per chiedere la lettura o una per la scrittura).

La figura 2 mostra l'interfaccia di sottosistema di memoria. Si noti che il bus di comunicazione fra la CPU e la memoria permette di comunicare con il MAR (comunicazione unidirezionale, dalla CPU alla memoria), con il MDR (comunicazione bidirezionale: dalla memoria alla CPU per lettura e dalla CPU alla memoria per scrittura); la figura inoltre evidenzia le linee di controllo che comunicano alla memoria se la CPU richieda di eseguire una lettura o una scrittura.

Le operazioni di memoria di lettura e scrittura vengono eseguite nel modo seguente.

Lettura

1. la CPU pone l'indirizzo di memoria richiesto nel MAR
2. la CPU segnala che vuole effettuare una lettura sull'opportuna linea di controllo

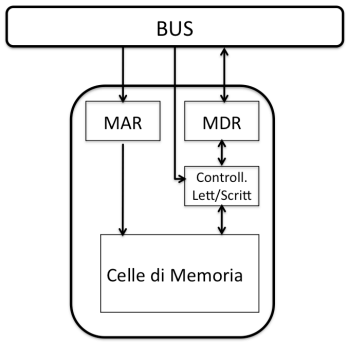

Figure 2: Il sistema di memoria

3. dopo aver atteso il tempo necessario alla memoria per l'operazione di lettura la CPU recupera il valore presente in MDR.

Scrittura

1. la CPU pone i nuovi contenuti che devono essere memorizzati in MDR
2. la CPU mette l'indirizzo di memoria richiesto nel MAR
3. la CPU segnala che vuole effettuare una scrittura sull'opportuna linea di controllo[5]

2.2 L'unità centrale di elaborazione e il ciclo di esecuzione di un'istruzione

La CPU è divisa in due parti principali:

- l'unità di controllo Control Unit (UC)
- l'unità logico/aritmetica (ALU)

L'unità di controllo è responsabile per il ciclo di esecuzione delle istruzioni, la ALU esegue le operazioni richieste.

Ricordiamo il concetto di programma memorizzato: il programma e di dati del programma in esecuzione sono contenuti nella memoria del computer, Inoltre, abbiamo visto che un programma è una sequenza di istruzioni. La sua esecuzione comporta l'esecuzione di una istruzione alla volta fino a quando non viene eseguita una speciale istruzione HALT che segnala la fine dell'esecuzione del programma e l'arresto della CPU.

La CPU per eseguire un'istruzione esegue ripete ciclicamente i seguenti passi:

1. **Fetch**, la fase in cui si reperisce in memoria la prossima istruzione da eseguire
2. **Decodifica dell'istruzione**, in cui la CPU analizza l'istruzione e decide che cosa deve essere fatto
3. **Esecuzione dell'istruzione**, in cui si esegue l'istruzione e si predispone la CPU per l'esecuzione della istruzione successiva

[5]Si noti che la CPU non deve attendere dopo un'operazione di scrittura che la memoria completi l'operazione.

2.2.1 La fase di reperimento dell'istruzione (Fetch)

Nel descrivere l'architettura di una CPU si possono ignorare numerosi aspetti che caratterizzano le architetture odierne. Un aspetto che non può essere omesso è l'esistenza del **Contatore di Programma (Program Counter, PC)** che ha il solo compito di contenere l'indirizzo di memoria della prossima istruzione da prelevare dalla memoria e da eseguire. Il contatore di programma viene utilizzato all'inizio della fase di fetch. Un altro è il **Registro Istruzioni (Instruction Register, IR)**, che in ogni momento contiene l'istruzione che è eseguita. Questo registro viene aggiornato alla fine della fase di fetch ponendovi la nuova istruzione da eseguire.

Ricordiamo che la CPU ha un solo tipo di memoria direttamente accessibile, i registri. Il Registro Istruzioni e il Contatore di Programma sono due registri che fanno parte dell'unità centrale. Gli altri registri della CPU sono utilizzati per memorizzare i dati.

Ricordiamo inoltre che l'interfaccia al sottosistema di memoria consiste di:

- Il Memory Address Register (MAR)
- Il Memory Data Register (MDR)
- La linea di controllo che specifica l'operazione di lettura/scrittura.

Siamo ora pronti a descrivere i passi che permettono di prelevare la prossima istruzione da eseguire e porla nel IR. I passi che avvengono sotto il comando dell'Unità di Controllo sono i seguenti:

1. Copiare il contenuto del PC nel MAR (poniamo nel MAR l'indirizzo della prossima istruzione da eseguire)
2. Impostare la linea di controllo per comunicare alla memoria la richiesta di lettura
3. Aspettare il reperimento dell'istruzione da parte della memoria
4. Copiare il contenuto di MDR nel registro istruzioni IR

Osserviamo che il passo 3 è sempre necessario, perché nessun sistema di memoria stato ancora inventato che può fornire i dati richiesti in tempo zero. Può sembrare strano ma, questo trasferimento è uno degli aspetti critici più significativi nel determinare la velocità complessiva della macchina. Eseguito il passo 4 i registri dell'Unità di Controllo possono esaminare l'istruzione a piacimento.

Ecco la versione Python del ciclo di esecuzione di un'istruzione:

```
runbit = True
PC = valore-iniziale # indirizzo della prima istruzione da eseguire
while runbit:
IR = fetch (PC) # preleva la prossima istruzione
PC = PC + 1 # aggiorna PC per eseguire la prossima istruzione
info = decodifica (IR) # decodifica l'istruzione
esegui (IR, info) # esegui l'istruzione - l'esecuzione può modificare runbit
o PC
```

Il valore booleano `runbit` si spiega semplicemente: la maggior parte dei computer hanno una istruzione chiamata HALT, o in modo simile, il cui scopo di fermare la CPU e far

terminare in questo modo l'esecuzione del programma. Chiaramente nell'algoritmo precedente, se l'esecuzione dell'istruzione pone questo bit pari a falso allora il ciclo di esecuzione si interrompe.

2.2.2 La fase di decodifica

Nella fase di decodifica l'unità di controllo esamina l'istruzione e imposta il controllo appropriato per la sua esecuzione. Ci limitiamo ad osservare che la CPU analizza l'istruzione e abilita i circuiti appropriati nelle ALU per produrre il risultato (ad esempio la somma di due valori presenti nei registri della CPU).

Le istruzioni del linguaggio macchina sono molto diverse fra loro a seconda dei diversi processori. Non entriamo nei dettagli del formato delle istruzioni. Possiamo comunque dire che un'istruzione del linguaggio deve avere almeno due parti:

1. una parte che specifica che tipo di istruzione si tratta; i bit di questa parte costituiscono il codice operativo (opcode)

2. una parte che specifica dove trovare i dati per effettuare l'operazione.

2.2.3 La fase di esecuzione e l'unità aritmetico/logica (ALU)

La fase di esecuzione di una istruzione dipende ovviamente dal tipo di istruzione. Non entriamo nei dettagli e ci limitiamo ad osservare che le istruzioni possono essere classificate in quattro diverse categorie: *i)* le istruzioni che coinvolgono il trasferimento di dati fra la memoria e la CPU, *ii)* quelle di ingresso/uscita che prevedono il trasferimento di dati fra la memoria e l'esterno, *iii)* quelle che controllano l'esecuzione del programma e, infine, *iv)* quelle che elaborano le informazioni eseguendo operazioni sui dati.

L'unità di controllo è responsabile della gestione del ciclo di elaborazione dell'informazione; è, quindi, responsabile del controllo complessivo e predispone il trasferimento dei dati nella CPU dando gli opportuni comandi alla memoria e ai dispositivi di ingresso e uscita.

Istruzioni aritmetiche e logiche
Per queste istruzioni il codice operativo specifica la particolare operazione che si vuole eseguire (ad esempio addizione, sottrazione, AND,OR, moltiplicazione, radice quadrata ecc.), mentre la parte dati dell'istruzione specifica dove reperire i dati stessi. In molti casi i dati coinvolti dalle operazioni sono quelli memorizzati in uno dei registri interni della CPU e, quindi, in questi casi non abbiamo trasferimento di valori dalla memoria. L'unità logico aritmetica (ALU) è responsabile per l'esecuzione delle istruzioni che portano all'elaborazione dei dati. Le operazioni che la ALU è in grado di eseguire variano da elaboratore a elaboratore e includono in genere tutte le operazioni aritmetiche e le operazioni logiche di base. In molti casi si possono eseguire anche operazioni complesse come la radice quadrata.

L'esecuzione di un'operazione che coinvolge la ALU prevede che i dati dai registri interni coinvolti nella operazione siano trasferiti agli ingressi della ALU e che alla ALU sia notifcata la specifica operazione richiesta. Quindi l'unità di controllo aspetta il tempo sufficiente a che la ALU sia in grado di completare l'operazione e quindi prevede il trasferimento del risultato in uno dei registri interni della CPU.

Istruzioni di lettura e scrittura in memoria e inizializzazione dei registri
Per le istruzioni di lettura e scrittura in memoria il codice operativo specifica il tipo di

trasferimento (lettura o scrittura) fra la memoria e uno dei registri della CPU; a seconda del particolare linguaggio macchina possiamo avere un ampio repertorio per specificare la cella (o le celle) di memoria coinvolte. Ad esempio possiamo anche trasferire valori da un registro ad un altro. Inoltre è possibile assegnare ai registri o alle celle di memoria valori costanti che non sono presenti in memoria ma che sono specificati nell'istruzione. Questa ultima opzione permette di realizzare le istruzioni che inizializzano le variabili.

La parte dati dell'istruzione determina, a seconda dei casi, la cella di memoria, il registro o i registri coinvolti e i valori costanti specificati nell'istruzione. L'esecuzione di una istruzione di questo tipo prevede che l'unità di controllo interagisca con la memoria secondo le modalità che abbiamo visto nella fase di fetch. Ovviamente i registri coinvolti in questi casi non sono il contatore di programma (PC) o il registro istruzione (IR) ma i registri della CPU che contengono i dati (REG).

Istruzioni di input/output
Per queste istruzioni il codice operativo specifica il tipo di trasferimento mentre la parte dati specifica sia il dispositivo coinvolto nell'operazione che la locazione di memoria (o le locazioni di memoria) coinvolte.

L'esecuzione di una di queste istruzioni è simile al caso precedente ma con la significativa differenza che in questo caso i dettagli sono più complessi e li omettiamo. Ci limitiamo a ricordare che anche in questo caso i diversi comandi che prevedono il trasferimento dei dati sono eseguiti dall'unità di controllo ma che spesso l'esecuzione delle operazioni può richiedere l'esecuzione di un programma specifico.

Istruzioni per il controllo dell'esecuzione del programma
L'esecuzione di una istruzione appartenente ad una delle tre categorie precedenti prevede che l'istruzione successiva da eseguire sia quella che segue immediatamente in corso di esecuzione. In questo caso l'unità di controllo deve prevedere di aggiornare il contatore di programma incrementandone il valore di una unità. È chiaro però che programmi con queste sole istruzioni non sono sufficienti. Innanzitutto abbiamo bisogno di una istruzione che determini l'arresto dell'esecuzione del programma. Senza molta fantasia questa istruzione è comunemente denominata HALT. Ma la presenza dell'istruzione HALT come sola istruzione per il controllo dell'esecuzione del programma non permette di scrivere programmi interessanti (ad esempio non ci sarebbe la possibilità di scrivere un ciclo).

I linguaggi di programmazione sono quindi forniti di istruzioni che alterano l'ordine di esecuzione delle istruzioni stesse e che sono dette istruzioni di salto; in queste istruzioni la parte che specifica i dati per eseguire l'istruzione indica dove è reperibile la prossima istruzione da eseguire. Esistono molte istruzioni di salto nei linguaggi macchina, che possono essere classificate in due categorie: istruzioni di salto incondizionato e condizionato. Una istruzione di salto incondizionato specifica la prossima istruzione da eseguire; la sua esecuzione comporta quindi la modifica del contatore di programma (PC). In una istruzione di salto condizionato il salto è condizionato alla verifica di una condizione (specificata nell'istruzione stessa). Ad esempio si può prevedere il salto solo se il valore di un certo registro è zero.

3 Un semplice linguaggio macchina

Lo studio di una vera e propria architettura di una CPU e di un linguaggio macchina in uso oggi va oltre lo scopo di queste dispense, e per questa ragione di limitiamo a considerare un

semplice computer ipotetico che chiameremo ELMAF, che sta per (Esempio di Linguaggio Macchina Fondamenti). Prima di continuare, osserviamo che ELMAF non è un linguaggio esistente ma è una versione molto semplificata di un linguaggio macchina reale. Innanzitutto assumiamo che l'elaboratore che esegue il programma sia un elaboratore con celle da trentadue bit e una memoria di 2^{26} locazioni di memoria, indirizzate da 0 a $2^{26} - 1$.

Ciascuna istruzione è composta da due parti

- una parte detta *codice operazione* che specifica l'operazione da eseguire
- una parte che specifica le informazioni necessarie alla sua esecuzione; vedremo nel seguito che queste informazioni specificano, ad esempio, il registro o i registri o una locazione di memoria coinvolti nell'istruzione.

Come abbiamo visto le istruzioni sono poste in memoria; nel seguito assumiamo che ciascuna istruzione occupi una singola cella di memoria[6]. Pertanto nel nostro caso abbiamo che tutta l'istruzione occupi 32 bit.

Presentiamo ora l'insieme di istruzioni che ELMAF può eseguire. Le operazioni sono specificate nella seguente tabella.

HALT - Arresta l'esecuzione del programma
LOAD X IND - Leggi cella di indirizzo IND e trasferisci il dato nel reg. X
LOADC X VALORE - Inizializza il registro X con VALORE
STORE X IND - Poni il valore del reg. X nella cella di indirizzo IND
ADD X,Y,Z - Esegui $X = Y + Z$, dove X, Y e Z sono registri
SUB X,Y,Z - Esegui $X = Y - Z$, dove X, Y e Z sono registri
MULT X,Y,Z - Esegui $X = Y * Z$, dove X, Y e Z sono registri
DIVIDE X,Y,Z - Esegui $X = Y/Z$, dove X, Y e Z sono registri
SQRT X,Y - Esegui $X = \sqrt{Y}$, dove X e Y sono registri
READ IND - Leggi un valore in ingresso e ponilo nella cella di indirizzo IND
WRITE IND - Scrivi il valore della cella IND sul dispositivo di uscita
JUMP S - Salta all'istruz. memorizzata in S ponendo PC pari a S
JUMPIFZERO X,S - Salta se il reg. X è zero ponendo il PC pari a S
JUMPIFNEG X,S - Salta se il reg. X è negativo ponendo PC pari a S

[6]In alcune architetture possiamo avere istruzioni che occupano più di una cella di memoria.

Poiché ELMAF prevede solo quattordici istruzioni è sufficiente utilizzare un codice binario di quattro bit per rappresentarle. In particolare possiamo associare il codice 0000 all'operazione HALT che comanda l'arresto del programma e via via valori successivi alle altre operazioni, il codice macchina dell'istruzione JUMPIFZERO sarà quindi 1110.

Il nostro computer molto semplice è dotato di sei registri interni: oltre al RI e al PC abbiamo altri quattro registri interni R1, R2, R3, R4 per memorizzare valori nella CPU. Per specificare uno fra quattro registri abbiamo bisogno quindi di due bit. Inoltre abbiamo assunto che la memoria abbia 2^{26} locazioni di memoria. Pertanto per indirizzare una cella di memoria è sufficiente specificare un indirizzo di ventisei bit.

Vediamo ora la semantica delle diverse istruzioni.

- Le istruzioni LOAD e LOADC permettono di assegnare valori ai registri. In particolare, la semantica delle istruzioni di LOAD prevede di specificare l'indirizzo di memoria da cui effettuiamo la lettura e il registro in cui carichiamo il dato. Ad esempio,

  LOAD R2 800

  specifica la memorizzazione il valore della locazione di memoria 800 nel registro 2. La sua esecuzione non richiede l'intervento della ALU ma solo del sottosistema memoria. Osserviamo che dei trentadue bit dell'istruzione quattro sono quelli che specificano il codice dell'istruzione, due quelli che e ventisei sono quelli che specificano l'indirizzo di memoria per un totale di trentadue bit, tanti quanto sono i bit di ciascuna cella di memoria.

- Invece la semantica dell'istruzione LOADC inizializza il registro con il valore specificato nell'istruzione. Ad esempio

  LOADC R2 800

  specifica la memorizzazione del valore 800 nel registro 2. Come nel caso dell'istruzione LOAD la sua esecuzione non richiede l'intervento della ALU; inoltre dato che il valore con cui inizializzare il registro è specificato nell'istruzione stessa ed è quindi memorizzato nel RI, l'esecuzione dell'istruzione non richiede l'utilizzo del sottosistema di memoria. Osserviamo che abbiamo un limite al numero di bit del valore che possiamo specificare; infatti, dato che ogni istruzione può al massimo essere formata da trentadue bit e quattro bit sono utilizzati per specificare il codice dell'istruzione e due per specificare il registro coinvolto nell'istruzione, rimangono ventisei bit per specificare il valore da caricare nel registro. Questo valore è quello trasmesso dalla CPU al sistema di memoria e posto nel MDR.

- Il caso dell'operazione STORE è analogo. Ad esempio per caricare il contenuto del registro 4 nella locazione 100 di memoria dobbiamo scrivere

  STORE R4 100

- Le istruzioni di ADD, SUB, MULT, DIV sono le operazioni che coinvolgono la ALU e specificano l'esecuzione delle operazioni aritmetiche di addizione, sottrazione, moltiplicazione e divisione e coinvolgono tre registri; gli ultimi due registri specificano gli operandi mentre il primo il risultato. Ad esempio

  ADD R1,R2,R3
  comporta la somme dei contenuti dei registri 2 e 3 e la memorizzazione del risultato nel registro 1. Invece

ADD R1,R1,R2
comporta la somma dei registri R1 e R2 e la memorizzazione del risultato in R1. La semantica delle istruzioni SUB,MULT,DIV è analoga. L'unica differenza consiste nel fatto che invece di sommare eseguiamo altre operazioni. In ogni caso il primo registro specifica dove memorizziamo il risultato mentre il secondo e il terzo specificano il primo e il secondo operando. In particolare abbiamo quindi che

SUB R1, R2, R3
esegue la differenza del valore memorizzato in R2 meno il valore memorizzato in R3 e pone il risultato in R1.

- L'istruzione SQRT esegue la radice quadrata di un numero. In particolare

  SQRT R1, R2
  esegue la radice quadrata del contenuto del registro R2 e pone il risultato in R1; osserviamo che l'operazione è corretta solo se il valore memorizzato in R2 è non negativo.

- L'istruzione di READ permette di immettere dall'esterno valori in una cella di memoria specificata nell'istruzione. Ad esempio l'esecuzione dell'istruzione

  READ 100
  comporta la lettura di un valore dal dispositivo di ingresso e la sua memorizzazione nella cella di memoria di indirizzo 100.

- L'istruzione di WRITE permette di fornire in uscita i risultati del calcolo utilizzando un dispositivo di uscita. Ad esempio, se il dispositivo di uscita è il video l'esecuzione dell'istruzione

  WRITE 200
  comporta la visualizzazione tramite un dispositivo di uscita del valore memorizzato nella cella 200.

Dal punto di vista dell'ordine di esecuzione delle istruzioni distinguiamo tre casi

1. istruzione HALT: questa istruzione prevede l'arresto del programma.

2. istruzione LOAD, STORE; ADD; SUB; MULT, DIV, SQRT, READ, WRITE: al termine dell'esecuzione, l'istruzione successiva da eseguire è l'istruzione che segue. Questo vuol dire che dopo aver eseguito l'istruzione il contatore di programma PC è incrementato di una unità.

3. istruzione JUMP (salto incondizionato): l'esecuzione di questa istruzione ha come unico scopo quelllo di specificare che la prossima istruzione da eseguire sia quella il cui indirizzo è specificato nell'istruzione. L'esecuzione di questa istruzione prevede quindi l'aggiornamento del PC al valore S specifcato nell'istruzione.

4. istruzioni JUMPIFZERO, JUMPIFNEG (salto condizionato): Per descrivere l'esecuzione dell'istruzione distinguiamo due casi. Se la condizione che è specificata nel test dell'istruzione è verificata allora il contatore di programma è aggiornato al valore specificato nell'istruzione; altrimenti il contatore di programma è incrementato di una unità in modo tale da proseguire l'esecuzione con l'istruzione successiva.

Osserviamo che per scrivere le istruzioni JUMP, JUMPIFZERO e JUMPIFNEG abbiamo bisogno di conoscere esattamente dove sia memorizzato il programma. Per tale aspetto facciamo riferimento al secondo esempio che segue.

Esempio 1 *Vediamo ora un semplice programma che legge tre numeri, ne calcola la somma e finisce stampando il risultato*

```
READ 1000
READ 1001
READ 1002
LOAD R1 1000
LOAD R2 1001
LOAD R3 1002
ADD R4,R1,R2
ADD R4,R3,R4
STORE R4 1003
WRITE 1003
HALT
```

Le prime tre istruzioni leggono dal dispositivo di ingresso i tre valori e li memorizzano nelle celle 1000,1001,1002. Le seconde tre istruzioni memorizzano i tre valori nei registri R1,R2,R3. L'istruzione ADD R4,R1,R2 pone in R4 la somma dei valori contenuti in R1 e R2, mentre l'istruzione ADD R4,R3,R4 pone in R4 il valore finale. Infine le istruzioni STORE e WRITE memorizzano e stampano il risultato.

Per poter eseguire il programma è necessario conoscere dove è memorizzato il programma e specificare come inizializzare il contatore di programma. Infatti dobbiamo essere in grado di poter inizializzare il contatore di programma all'indirizzo di memoria che contiene la prima istruzione del programma. Non entreremo nei dettagli di come questa operazione è realizzata oggi; nel passato era eseguita tramite un pannello di controllo in cui l'operatore fisicamente inizializzava il contatore di programma[7].

Si noti che dopo avere eseguito ciascuna istruzione del programma il contatore di programma viene aggiornato di una unità fino a quando dopo aver eseguito l'istruzione HALT il programma si arresta.

Esempio 2 *Programma che risolve un'equazione di secondo grado. Data un'equazione di secondo grado* $ax^2 + bx + c = 0$ *vogliamo calcolarne le radici.*
In questo caso assumiamo che all'inizo del programma le locazioni di memoria 100, 101, 102 contengano le costanti a, b, c*. Ricordiamo che l'equazione ha soluzione solo se* $b^2 - 4ac$ *è non negativo.*

Il programma seguente prima calcola $b^2 - 4ac$*; se questo valore è negativo allora viene eseguita un'istruzione di salto che porta alla terminazione del programma senza stampare nulla. Altrimenti vengono calcolate le due soluzioni.*

L'osservazione cruciale è che per stabilire l'istruzione di salto dobbiamo conoscere esattamente dove è memorizzato il programma. In questo caso assumiamo che il programma sia

[7] I computer utilizzati all'inizio erano dotati di un pannello principale di luci. Quando questi computer esguivano programmi, le luci davano informazioni che permettevano ad un osservatore esperto di sapere cosa stava accadendo all'interno del sistema in quel momento.

memorizzato a partire dalla locazione di memoria 1000, una istruzione in ogni locazione di memoria. Osserviamo quindi che l'istruzione HALT è memorizzata nella locazione 1023.

LOAD R1 100 (le prime tre istruzioni caricano i coefficienti a, b, c nei tre registri R1 R2 R3)
LOAD R2 101
LOAD R3 102
LOADC R4 4 (pone la costante 4 nel registro R4)
MULT R2 R2 R2
MULT R4 R3 R4
MULT R4 R1 R4 (a questo punto R4 contiene $4ac$)
SUB R3 R2 R4 (a questo punto R3 contiene $b^2 - 4ac$)
JUMPIFNEG R3 1023 (se $b^2 - 4ac < 0$ non esistono soluzioni e il programma termina)
SQRT R4 R4 (a questo punto R4 contiene $\sqrt{b^2 - 4ac}$)
LOADC R2 0
LOAD R3 101
SUB R2 R2 R3 (ora R2 contiene $-b$)
LOADC R3 2
MULT R1 R3 R1 (pone in R1 il valore $2a$)
ADD R3 R2 R4 (a questo punto R3 contiene $-b + \sqrt{b^2 - 4ac}$)
DIV R3 R3 R1 (a questo punto R3 contiene $\frac{-b+\sqrt{b^2-4ac}}{2a}$)
STORE R3 103 (memorizza in 103 la prima soluzione trovata)
SUB R3 R2 R4 (a questo punto R3 contiene $-b - \sqrt{b^2 - 4ac}$)
DIV R3 R3 R1 (a questo punto R3 contiene $\frac{-b-\sqrt{b^2-4ac}}{2a}$)
STORE R3 104 (memorizza in 104 la seconda soluzione trovata)
WRITE 103 (questa istruzione e la successiva stampano i risultati)
WRITE 104
HALT

Osserviamo che l'istruzione di salto specifica il salto all'istruzione memorizzata nella locazione 1023 perché 1023 è la locazione di memoria che memorizza l'istruzione HALT. Se avessimo memorizzato il programma in una locazione di memoria diversa avremmo dovuto riscrivere il programma. Senza dubbio questa è una ulteriore complicazione perché richiede al programmatore di specificare al computer dove memorizzare il suo programma. Fortunatamente il problema precedente è risolto nei calcolatori moderni con metodi automatici che non richiedono l'intervento del programmatore.

L'esame dei programmi in linguaggio macchina precedenti evidenzia quanto abbiamo già osservato: la scrittura di programmi in linguaggio macchina è lunga e noiosa. Si confronti ad esempio il programma precedente per il calcolo delle soluzioni di una equazione di secondo grado con l'equivalente programma Python, riportato nel seguito. Questo aspetto rende l'attività di programmazione più complessa e facilita inoltre la possibilità di fare errori.

Programma Python per la soluzione di un'equazione di secondo grado

```
import math
a=1
b=3
c=1
```

```
d=b*b-4*a*c
if d>0:
   s1=(-b+math.sqrt(d))/(2*a)
   s2=(-b-math.sqrt(d))/(2*a)
   print(s1, s2)
else:
   print('soluzioni complesse')
```

La seconda osservazione riguarda il fatto che la CPU non distingue tra una locazione di memoria che contiene istruzioni del programma e una che contiene dati. In particolare potremmo scrivere un programma che si automodifica! Infatti potremmo scrivere un programma che contiene una istruzione che modifica il contenuto di una cella di memoria che contiene una istruzione stessa del programma. Chiaramente l'analisi e la scrittura di programmi di questo tipo è molto complicato e - in genere - da evitare. Non daremo esempi di questo tipo di programmi e ci limitiamo ad osservare che spesso virus e altri software maliziosi che attaccano e rendono inutilizzabile un calcolatore utilizzano questa possibilità di modificare locazioni di memoria che contengono programmi.

4 La legge di Moore

La legge prende il nome dal co-fondatore di Intel, Gordon E. Moore, che ha descritto l'andamento in un articolo pubblicato nel 1965 partendo dall'osservazione che il numero di componenti a circuiti integrati era raddoppiato ogni anno dalla invenzione del circuito integrato nel 1958 fino al 1965; Moore prevedeva che tale fenomeno sarebbe proseguito "per almeno dieci anni". La previsione si è dimostrata incredibilmente accurata, e la legge è ora utilizzata nel settore dei semiconduttori per guidare la pianificazione a lungo termine e fissare obiettivi di ricerca e sviluppo. Si fa spesso riferimento anche ad una variante della legge, dovuta a David House, che prevede un raddoppio delle prestazioni dei chip (le prestazioni sono una combinazione dell'effetto di più transistor che permette di avere chip più potenti e della velocità dei chip) ogni diciotto mesi.

Le funzionalità di molti dispositivi elettronici digitali sono fortemente legati alla legge di Moore: velocità di elaborazione, capacità di memoria, e anche il numero e la dimensione dei pixel nelle fotocamere digitali. Tutti questi parametri sono migliorati a tassi esponenziali. Questo miglioramento esponenziale ha migliorato notevolmente l'impatto dell'elettronica digitale in quasi tutti i segmenti dell'economia mondiale. Possiamo pertanto affermare che la legge di Moore descrive un aspetto cruciale del cambiamento tecnologico e sociale degli ultimi cinquant'anni.

Infatti la figura che descrive la densità dei dispostivi in funzione del tempo è in scala logaritmica; quindi, in particolare, un raddoppio delle prestazioni ogni diciotto mesi implica che ogni tre anni le prestazioni quadruplicano, e che aumentano di dieci volte in meno di cinque anni di mille volte in quindici anni e così via[8].

Anche se la previsione di Moore è valida da più di mezzo secolo, essa deve essere considerata una osservazione empirica e non una legge fisica o naturale. Infatti la tendenza odierna prevede un rallentamento e, in particolare, il raddoppio della densità solo ogni tre anni.

[8]Per avere un'idea dell'enorme miglioramento tecnologico si può immaginare che se l'industria automobilistica fosse avanzata negli ultimi cinquanta anni come l'industria dei semiconduttori una Rolls Royce oggi consumerebbe un litro di benzina ogni duecentomila chilometri e costerebbe meno del costo del parcheggio.

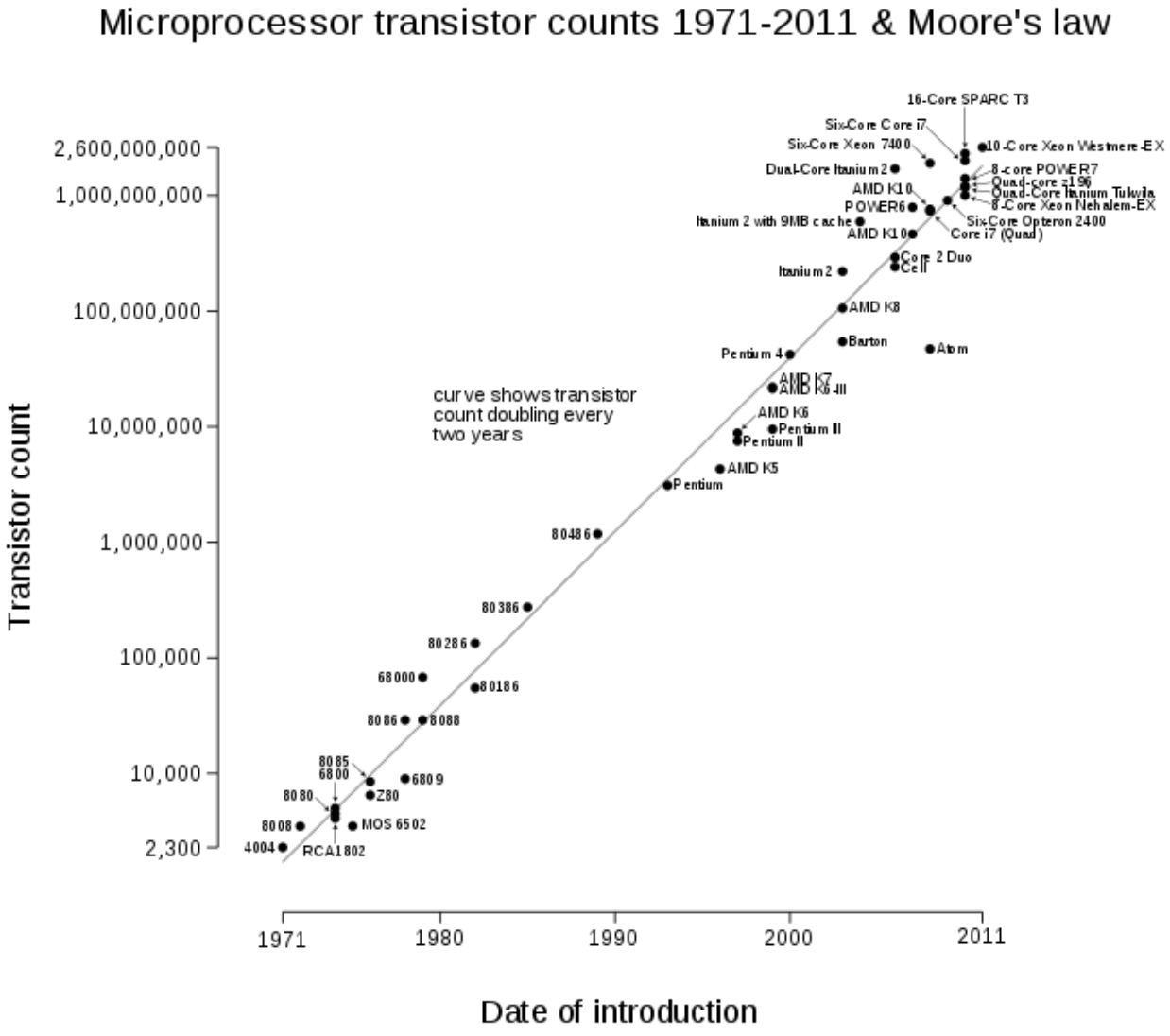

Figure 3: La Legge di Moore nel 1971-2011 (immagine tratta da Wikipedia, Wikimedia commons).

Moore ha formulato al sua legge con riferimento alla densità dei componenti (transistor). Osserviamo però che diverse misure della tecnologia digitale stanno migliorando a tassi esponenziali simili a quelli indicati da Moore come, ad esempio, le dimensioni, il costo, la densità e la velocità dei componenti.

Densità dei componenti. Nel 1971 una piccola compagnia chiamata Intel ha realizzato il 4004 il suo primo microprocessore. Il chip, un quadrato di 12 millimetri conteneva 2300 transistors - gli elementi base dei computer digitali. La distanza fra due transistor vicini era 10.000 nanometri, approssimativamente le dimensioni della cellula di un globulo rosso. Il 4004 era un grande successo di miniaturizzazione ma vicino a una scala umana: un buon telescopio di uso casalingo avrebbe permesso di distinguere i diversi transistors.

Per avere una idea dello sviluppo osserviamo che lo sviluppo tecnologico ha consentito dimensioni minime nella produzione di chip con le dimensioni del chip prodotto da Intel (nominato Broadwell) di 14 nanometri. Questo valore è molto molto piccolo (si pensi che le dimensioni di un atomo dell'ordine di $0,1$ nanometri).

La legge di Moore stabilisce un miglioramento esponenziale del numero di dispositivi che è possibile mettere su un chip e quindi realizzare su un singolo chip elaboratori sempre più potenti. Infatti gli elaboratori realizzati nel tempo erano all'inizio dotati di istruzioni sempre più potenti. Successivamente a partire dagli anni ottanta del secolo scorso l'aumento del numero di circuiti ha permesso di realizzare CPU in grado di eseguire più istruzioni contemporaneamente. I continui miglioramenti relativi alla densità di circuiti su un chip ha permessi che le CPU che si trovano oggi sul mercato contengono più di un unità centrale di elaborazioni (i cosiddetti chip multicore).

Velocità della CPU e tempi di clock. Un altro aspetto collegato direttamente alla dimensione dei dispostivi è la velocità della CPU che si misura nella frequenza di **clock**. In elettronica il termine clock indica un segnale periodico, generalmente un'onda quadra, utilizzato per sincronizzare il funzionamento dei dispositivi elettronici digitali [9].

La frequenza di clock si esprime in cicli al secondo, o hertz, e suoi multipli. L'unità di calcolo del calcolatore Z1 (che utilizzava la tecnologia elettromeccanica, non elettronica) costruito dal tedesco Konrad Zuse nel 1938 aveva una velocità massima di circa 1 hertz, che scriviamo anche come 1 Hz, eseguiva quindi un ciclo al secondo. La velocità di clock del primo microprocessore moderno, l'Intel 4004 progettato dall'italiano Federico Faggin nel 1971, era di 740 kHz. Le attuali CPU dei personal computer raggiungono i 3 Giga hertz (pari quindi a 3 miliardi di cicli al secondo).

Per capire come il tempo di clock influenzi la velocità di esecuzione di un programma osserviamo che una CPU è composta da vari circuiti preposti ad effettuare operazioni diverse, che interagiscono tra loro scambiandosi informazioni al fine di eseguire le istruzioni macchina. Affinché la sequenza di operazioni che realizza un'istruzione avvenga correttamente però è necessario che ad ogni circuito sia indicato il momento esatto in cui sono validi i segnali che riceve in ingresso. Il clock è il segnale che si occupa di questo ed è condiviso tra tutti i circuiti presenti culla CPU. La sua frequenza quindi deve essere calcolata in modo tale che il ciclo di clock sia sufficientemente lungo da consentire a tutti i circuiti, anche il più lento, di completare correttamente le proprie operazioni. Senza entrare nei dettagli architetturali

[9] In particolare, la velocità di una CPU viene misurata come la frequenza di clock; la velocità di clock dipende dal numero di commutazioni tra i due livelli logici "0" e "1" che circuiti logici interni ad un processore sono in grado di eseguire in un secondo

di un elaboratore possiamo dire che l'esecuzione di un'istruzione macchina richiede uno o più cicli di clock a seconda della istruzione macchina.

La discussione precedente spiega come maggiore sia di clock la frequenza, più sono le operazioni che possono essere eseguite nello stesso intervallo di tempo. Un incremento esponenziale dei tempi di clock si è avuto dagli inizi dei calcolatori fino al 2002-2004 circa. Successivamente è iniziato un rallentamento nell'incremento; osserviamo infatti che la frequenza di clock odierne sono simili a quelle di circa dieci anni fa[10].

Costo della memoria. Una legge simile vale per il costo di memorizzazione su disco per unità di informazione. In realtà il tasso di progressione di memorizzazione su disco negli ultimi decenni ha effettivamente accelerato più di una volta, che corrisponde all'introduzione di nuove tecnologie e di nuovi codici. L'attuale tasso di incremento della capacità del disco rigido è più o meno simile al tasso di incremento del numero di transistor[11].

Capacità di rete. Una legge analoga afferma che la quantità di dati trasmessi da una fibra ottica sta raddoppiando ogni nove mesi. Così, il costo di trasmissione di un bit su una rete ottica diminuisce della metà ogni nove mesi. Questo miglioramento è stato realizzato attraverso la possibilità di multiplazione a divisione di lunghezza d'onda che permette di trasmettere più informazioni utilizzando lunghezze d'onda diverse sulla stessa fibra. In questo modo aumenta la capacità di trasmissione di una fibra può aumentare per un fattore di 100.

Costo pixel. L'elemento unitario per la rappresentazione grafica è detto "pixel" e rappresenta un punto illuminato in uno schermo grafico. Maggiore è il numero di pixel per centimetro quadrato migliore è la qualità dell'immagine. Analogamente ai casi precedenti se consideriamo il costo di un "pixel per dollaro" come misura di base di valore per una fotocamera digitale, possiamo osservare un andamento esponenziale.

Costo semplici dispositivi. Poiché i processori presentati aumentano di potenza essenzialmente a parità di prezzo, ne consegue che i processori della generazione precedente, la cui potenza rimane fissa, calano di prezzo. Questo ha permesso la pervasività dell'informatica in molte attività e processi industriali.

La seconda legge di Moore. Ovviamente, per incrementare le prestazioni, occorrono sempre maggiori studi, ricerche e test; questo vuol dire un significativo costo di investimento. Per aumentare il numero di transistor all'interno del processore, senza aumentare la dimensione del processore stesso, occorrono dei componenti sempre più piccoli, quindi attività di ricerca su nuovi materiali che permettano miglioramenti significativi. L'aumento delle prestazioni comporta dei test, sia per provare la resistenza dei materiali, sia per l'affidabilità stessa del

[10] Ad esempio il primo computer MacIntosh della Apple, uscito nel 1984 aveva un clock di 8M Hertz, il secondo processore Pentium della Intel, prodotto nel 1994 aveva una frequenza di clock di tra 75 e 120 M Hertz a seconda delle versioni. Successivamente un nuovo approccio al progetto della CPU ha permesso di incrementare significativamente il tempo di clock; ad esempio il processore Pentium 3 della Intel, presentato nel 1999 aveva una velocitdi di 500-600 M Hertz a seconda delle versioni; si noti quindi un miglioramento della velocità per un fattore cinque in soli cinque anni. Successivamente abbiamo iniziato ad osservare un rallentamento: il Pentium M presentato nel 2004 aveva una frequenza di clock compresa fra 1,4 G Herz e 2,1 G Herz a seconda delle versioni, un miglioramento di "solo" due volte in cinque anni.

[11] Per dare un'idea concreta uno dei primi personal computer Dell, presentato sul mercato nel 1990, il Dell System 425E aveva una memoria centrale di 4 Mbit e costava circa ottomila dollari. Oggi un personal computer di media potenza ha una memoria di 4Giga bit, mille volte più grande e costa dieci volte di meno.

processore. Tutto questo ovviamente comporta delle spese che la casa produttrice deve affrontare se vuol avere un prodotto innovativo. Questi costi sono sia di ricerca e sviluppo ma anche costi di nuove tecnologie. Per questa ragione i costi di produzione sono aumentati costantemente a ogni nuova generazione di processori: la *seconda legge di Moore*, afferma che il costo del capitale di una fabbrica di semiconduttori per realizzare una nuova generazione di processori aumenta esponenzialmente nel tempo.

Al momento attuale, tenuto conto che l'entità dell'investimento dipende in maniera significativa dal tipo di prodotto in sviluppo e dalle economie di scala che si intendono effettuare, una stima stabilisce intorno ai 2-4 miliardi di euro il costo per progettare e realizzare una fabbrica per una nuova generazione di chip. Questi costi molto alti implicano un fenomeno di netto consolidamento del settore, sintomo di una industria matura, con poche aziende di grandi dimensioni, con alte barriere di ingresso di nuovi produttori, tendenza sensibile all'oligopolio ed una forte riduzione della propensione al rischio.

Osserviamo però che l'aumento delle capacità di calcolo non sempre permette una corrispondente riduzione dei tempi di esecuzione dei programmi. Infatti spesso è stato osservato che le versioni successive di un programma sono sensibilmente più lente, utilizzano più memoria e potenza di elaborazione, oppure hanno requisiti hardware più elevati rispetto alla versione precedente, apportandovi solo limitati miglioramenti percepibili dall'utente. Il termine inglese "software bloat" indica questo fenomeno (letteralmente "gonfiamento del software"), spesso usato come dispregiativo dagli utenti finali per descrivere indesiderati cambiamenti di interfaccia utente anche se questi cambiamenti hanno avuto poco o nessun effetto sui requisiti hardware ma che comportano maggiori tempi di esecuzione. Il termine software bloat è stato introdotto nel 2008 esaminando le versioni successive di Microsoft Office tra il 2000 e il 2007 (Microsoft Office è il programma di Microsoft per la scrittura e la formattazione di testi). Nonostante i guadagni di prestazioni computazionali durante questo periodo di tempo avvenuti secondo la legge di Moore, Office 2007 su un computer del 2007 ha eseguito la stessa operazione richiedendo circa il doppio del tempo rispetto a Office 2000 su un computer dell'anno 2000.

Il fenomeno può verificarsi a causa di una scarsa attenzione all'efficienza del software a favore di altre preoccupazioni come la produttività degli sviluppatori, o, aspetti relativi alla sicurezza che non sono immediatamente percepiti dagli utenti ma che comunque richiedono programmi più complessi e, quindi, con maggiori tempi di esecuzione.

4.1 I limiti estremi della legge di Moore

I limiti della prima legge di Moore potrebbero essere solo quelli stabiliti dal raggiungimento dei limiti fisici imposti per la riduzione delle dimensioni dei transistor, e quindi della scala di integrazione, al di sotto dei quali si genererebbero effetti 'parassiti' indesiderati di natura quantistica nei circuiti elettronici. Tali limiti sono molto vicini con le ultime generazioni di processori per cui l'unico modo possibile e praticabile per aumentare le prestazioni di calcolo e processamento dati è rappresentato dalla tecnologia multicore (che prevede la realizzazione di più processori sullo stesso chip) oppure l'affiancamento di molti calcolatori in parallelo come avviene peraltro nei supercalcolatori dei centri di calcolo. La domanda attuale è per quanto tempo avremo miglioramenti nel futuro. Nel 2005 Gordon Moore ha dichiarato in un'intervista che la legge non può continuare per sempre per motivi tecnologici e fisici. Egli ha anche notato che se la legge continuasse nel tempo i transistor potrebbero raggiungere i limiti della miniaturizzazione a livello atomico intorno al 2025.

Questi valori pongono nuove sfide. In particolare osserviamo che Intel ha annunciato per il 2016 una tecnologia da 10 nanometri e prevede di arrivare prima del 2020 a 5 nanometri. Questi valori sono molto molto piccoli e questi obiettivi sono stati posticipati: Intel prevede l'introduzione dei chip a 10 nanometri per la fine del 2017. Per avere idea della difficoltà di ulteriori miniaturizzazioni ricordiamo che un atomo ha le dimensioni di circa $0,1$ nanometri e quindi una tecnologia da 10 nanometri realizza un dispositivo elettronico con circa 100 atomi.

Analoghe considerazioni valgono per i tempi di clock. Abbiamo già osservato che negli ultimi anni abbiamo avuto un rallentamento della legge di Moore per quanto riguarda la frequenza di clock. I motivi di questo fenomeno sono da individuare nel fatto che tempi di clock come quelli attuali sono vicini al limite fisico della fisica classica, come suggerisce il seguente ragionamento.

Un processore che opera a 3 G Herz ha un tempo di clock pari a $1/(3 * 10^9)$ secondi, pari quindi a $0,33$ nanosecondi. Non è questa la sede per entrare nei dettagli di cosa viene eseguito da una CPU in un tempo di clock; ci basti sapere che in questo tempo la CPU esegue operazioni elementari, come ad esempio la somma di due numeri.

Sappiamo che inoltre che luce viaggia ad una velocità di 300.000 Km al secondo (pari a $3 * 10^{11}$ centimetri al secondo) e che questo limite è insuperabile. Pertanto anche i segnali elettrici non possono viaggiare ad una velocità superiore ad essa. Un semplice calcolo ci permette di verificare che la distanza percorsa dalla luce in un tempo pari a $0,33$ nanosecondi è pari a circa 11 centimetri, circa tre volte più grande della dimensione di un chip attuale. Possiamo quindi concludere che progettare un chip delle dimensioni di un centimetro quadrato, che sia in grado di operare con una frequenza di clock superiore a 30 G Herz dovrebbe essere in grado di eseguire un'operazione nel tempo che la luce impiega ad attraversare il chip stesso. Questo sembra pertanto un limite al momento invalicabile.

Un secondo limite intrinseco della tecnologia attuale è il limite termico. Non entriamo nei dettagli di una trattazione fiisca e ci limitiamo ad osservare che ogni cambiamento di stato di un singolo transistor (che realizza un semplice ad esempio la memorizzazione di un bit) richiedono una quantità di energia finita. Questa quantità è molto piccola ma è diversa da zero e dissipandosi produce calore.

Il limite allo sviluppo tecnologico è dato dal fatto che al crescere della velocità e del numero di componenti su ogni singolo chip la quantità di calore prodotta cresce. L'aumento di calore produce un innalzamento della temperatura del chip tale da non permetterne il funzionamento neanche con i più sofisticati sistemi di raffreddamento. In particolare osserviamo che il sistema di raffreddamento di un chip è oggi molto grande e se rimosso porta in poco tempo alla fusione del chip. Si veda ad esempio il flimato del 2006 `https://www.youtube.com/watch?v=Xf0VuRG7MN4` che fa riferimento a chip in uso prodotti nel periodo 2000-2006.

Infatti, la quantità di calore prodotta da un chip di una data dimensione è una funzione del tipo *Calore*$= a \times f \times n$ dove a è una costante, f è la frequenza a cui opera il chip e n è il numero di componenti per centimetro quadrato. In altre parole la quantità di calore è tanto maggiore quanto maggiore è la frequenza f (infatti maggiore è la frequenza a cui un circuito opera più spesso i singoli componenti cambiano stato e, quindi, maggiore è il calore sviluppato) e dal numero di dispositivi che esso contiene (infatti la quantità di energia sviluppata dipende dal cambiamento di stato di ciascun componente). Il limite intrinseco è dato dal fatto che al crescere di f e n il calore sviluppato dal circuito alzerebbe la temperatura fino a fondere il circuito stesso indipendentemente dal sistema di raffreddamento usato.

Concludiamo osservando che le dimensioni e la velocità dei dispositivi attuali sono vicini ai limiti fisici e pongono problemi di fisica quantistica che finora sono stati ignorati nella

progettazione dei processori. Futuri miglioramenti che permettono di mantenere la legge di Moore nel futuro nel lungo termine richiedono nuove idee e nuovi approcci. Ad esempio, secondo molti ricercatori potremmo superare i limiti fisici progettando calcolatori che sfruttano le leggi della fisica quantistica. Gli studi teorici mostrano che effettivamente in questo modo saremmo in grado di avere computer molto potenti; sfortunatamente realizzare un elaboratore quantistico è al momento un compito formidabile che non sappiamo come risolvere: gli elaboratori quantistici che sono stati realizzati nei laboratori sono al momento costosi e molto limitati nelle capacità di calcolo.

4.2 Conclusioni

In un mondo dove tante cose e prodotti industriali hanno da tempo miglioramenti limitati ogni dieci anni (si pensi ad esempio alla velocità di crociera degli aerei di linea o delle automobili, o alle altezze degli edifici) la crescita esponenziale della potenza di calcolo e la riduzione dei costi dei computer che abbiamo osservato negli ultimi cinquanta anni sono da considerarsi eccezionali e hanno radicalmente mutato la nostra vita quotidiana.

Questo progresso è avvenuto sfruttando la tecnologia dei circuiti integrati e oggi osserviamo difficoltà nuove che portano ad un rallentamento della velocità di progresso e aumenti di costi. Infatti le attuali soluzioni sono non lontane dai limiti fisici e pertanto non sappiamo se la legge di Moore continuerà realizzando lo stesso miglioramento che abbiamo osservato in questi anni, e per quanto tempo continuerà. La discussione in tal senso è aperta e approcci totalmente differenti sono allo studio.

In particolare, in questi anni sono apparse proposte radicali che offrono qualche speranza per ridefinire il computer stesso. Un'idea è quella di sfruttare la meccanica quantistica per eseguire calcoli molto più velocemente rispetto a quello che qualsiasi computer classico potrebbe mai sperare di fare. Un altro approccio cerca di emulare il cervello biologico, che sappiamo è in grado di realizzare operazioni complesse. Questi approcci sono ancora in fase sperimentale di laboratorio e non è al momento chiaro se e quando porteranno a prodotti sul mercato

La linea che finora appare più promettente è quella di diffondere il potere del computer piuttosto di concentrarlo in un unico dispositivo, diffondendo la capacità di calcolare e comunicare attraverso una sempre più ampia gamma di piccoli computer distribuiti. Ad esempio l'Internet degli oggetti (Internet Of Things -IOT) prevede di associare agli oggetti di uso quotidiano un dispositivo in grado di calcolare e comunicare con altri dispositivi.