David A. Patterson John L. Hennessy

Struttura e progetto dei calcolatori

Progettare con RISC-V

Edizione italiana a cura di Alberto Borghese

INFORMATICA ZANICHELLI

David A. Patterson John L. Hennessy

Struttura e progetto dei calcolatori

Progettare con RISC-V

Edizione italiana a cura di Alberto Borghese

Se vuoi accedere alle risorse online riservate

1. Vai su **my.zanichelli.it**
2. Clicca su *Registrati*.
3. Scegli *Studente*.
4. Segui i passaggi richiesti per la registrazione.
5. Riceverai un'email: clicca sul link per completare la registrazione.
6. Cerca la tua chiave di attivazione stampata in verticale sul bollino argentato in questa pagina.
7. Inseriscila nella tua area personale su **my.zanichelli.it**

Se sei già registrato, per accedere ai contenuti riservati di altri volumi ti serve solo la relativa chiave di attivazione.

INFORMATICA ZANICHELLI

Indice generale

Prefazione

La cosa più bella che possiamo sperimentare è il mistero;
esso è la fonte della vera arte e della vera scienza.

Albert Einstein, *What I Believe*, 1930

Guida al libro

Crediamo che lo studio dell'informatica e dell'ingegneria informatica debba non solo riguardare i principi su cui è fondata l'elaborazione, ma anche riflettere lo stato delle conoscenze attuali in questi campi. Crediamo anche che i lettori, qualunque sia la branca dell'informatica nella quale lavorano, possano apprezzare i paradigmi secondo i quali sono organizzati i sistemi di elaborazione, che determinano le loro funzionalità e prestazioni, e ne decretano in ultima analisi il successo.

La tecnologia richiede oggi che i professionisti di tutte le branche dell'informatica conoscano sia il software sia l'hardware, la cui interazione a tutti i livelli è la chiave per capire i principi fondamentali dell'elaborazione. Inoltre, le idee che stanno alla base dell'organizzazione e della progettazione dei calcolatori valgono sia nell'ambito informatico sia in quello dell'ingegneria elettronica e sono le stesse sia che il vostro interesse principale sia il software sia che sia l'hardware. Per questo motivo, nel testo l'enfasi viene posta sulla relazione tra hardware e software e vengono approfonditi i concetti che stanno alla base dei calcolatori delle ultime generazioni.

Il passaggio recente dalle architetture uniprocessore ai multiprocessori multicore ha confermato quanto sia corretta questa prospettiva, che abbiamo adottato fin dalla prima edizione di questo libro. Fino a poco tempo fa i programmatori potevano fare affidamento sul lavoro dei progettisti delle architetture e dei compilatori e dei produttori dei chip, per rendere più veloci o più efficienti dal punto di vista energetico i propri programmi senza il bisogno di apportare alcuna modifica. Questa era è finita: affinché un programma possa essere eseguito più velocemente, deve diventare un programma parallelo. Anche se l'obiettivo di molti ricercatori è fare sì che i programmatori non si accorgano della natura parallela dell'hardware per il quale scrivono i loro programmi, ci vorranno molti anni prima che ciò divenga effettivamente possibile. Crediamo

che nel prossimo decennio la maggior parte dei programmatori dovrà capire a fondo il legame tra hardware e software perché i programmi vengano eseguiti in modo efficiente sui calcolatori paralleli.

Questo libro è rivolto principalmente a coloro che, pur avendo scarse conoscenze del linguaggio assembler e della logica digitale, vogliono capire i concetti di base dell'organizzazione degli elaboratori, e a quei lettori che sono interessati a capire il modo in cui si progetta un elaboratore, come funziona e perché si ottengono determinate prestazioni.

L'altro testo degli stessi autori sulle architetture

Alcuni lettori conosceranno il testo degli stessi autori *Computer Architecture: A Quantitative Approach*, chiamato anche "Hennessy Patterson", mentre questo libro viene solitamente chiamato "Patterson Hennessy". Avevamo scritto quel libro con l'obiettivo di descrivere i principi su cui sono basate le architetture degli elaboratori utilizzando un robusto approccio ingegneristico per illustrare tutti i compromessi tra costi e prestazioni che si rendono necessari. In quel libro avevamo utilizzato un metodo basato su esempi e misure, effettuate su architetture disponibili sul mercato, per consentire al lettore di fare esperienza con la progettazione di sistemi realistici: l'obiettivo era dimostrare che le architetture degli elaboratori possono essere studiate utilizzando metodologie quantitative invece di un approccio descrittivo. Quel libro era stato pensato per i professionisti coscienziosi che volevano approfondire nei dettagli il funzionamento dei calcolatori.

La maggior parte dei lettori di questo libro non ha intenzione di diventare un progettista di calcolatori. Tuttavia, le prestazioni e l'efficienza energetica dei sistemi software prodotti nel prossimo futuro saranno enormemente influenzate da quanto bene i progettisti software avranno capito le strutture hardware di base che sono presenti in un calcolatore. Perciò, i progettisti dei compilatori e dei sistemi operativi, così come i programmatori di database e della maggior parte delle applicazioni, hanno bisogno di conoscere bene i principi fondamentali in base ai quali funziona un calcolatore, presentati in questo libro. Analogamente, i progettisti hardware devono capire a fondo come i loro progetti andranno a influenzare le applicazioni software.

Ci siamo quindi resi conto che questo libro doveva essere molto di più di una semplice estensione del materiale contenuto nell'altro testo, per cui abbiamo riesaminato a fondo il contenuto e lo abbiamo modificato adattandolo ai lettori di questo libro. Il risultato ha avuto così tanto successo che abbiamo eliminato tutto il materiale introduttivo dalle versioni successive di *Computer Architecture* e, quindi, ora la sovrapposizione dei contenuti tra i due libri è minima.

Perché il RISC-V per questa edizione?

La scelta dell'architettura dell'insieme di istruzioni è chiaramente critica per un libro di testo sulle architetture degli elaboratori. Non volevamo un insieme di istruzioni che richiedesse la descrizione di caratteristiche non fondamentali e barocche per chi si avvicina per la prima volta alle architetture, per quanto l'insieme di istruzioni sia popolare. Idealmente, il primo insieme di istruzioni dovrebbe fungere da modello, più o meno come il primo amore: sorprendentemente si ricorderanno entrambi con passione.

Dato che ci sono così tante possibili scelte oggigiorno, per la prima edizione del nostro testo *Computer Architecture: A Quantitative Approach*, abbiamo inventato il nostro personale insieme di istruzioni in stile RISC. Data la popolarità crescente, l'eleganza e la semplicità dell'insieme delle istruzioni MIPS siamo passati a utilizzare i processori MIPS per la prima edizione di questo libro di

testo e per le edizioni successive dell'altro libro. Il MIPS è stato di grande utilità per noi e i nostri lettori.

Sono trascorsi 20 anni da quando siamo passati al MIPS, e anche se miliardi di chip contenenti processori MIPS vengono ancora prodotti, questi si trovano tipicamente inseriti (*embedded*) in dispositivi dove l'insieme delle istruzioni è praticamente invisibile. E, quindi, da un po' di tempo è diventato difficile trovare un calcolatore reale sul quale i lettori possano scaricare un programma MIPS ed eseguirlo.

La buona notizia è che un insieme di istruzioni pubblico che aderisce da vicino ai principi RISC ha fatto il suo debutto e sta rapidamente guadagnando seguaci. Il RISC-V, che è stato sviluppato inizialmente all'Università di Berkeley, non solo ripulisce le stranezze dell'insieme delle istruzioni MIPS, ma offre un semplice, elegante e moderno esempio di quello a cui dovrebbe assomigliare un insieme di istruzioni nel 2017.

Inoltre, dato che non è un'architettura proprietaria, esistono simulatori, compilatori e debugger RISC-V "open source" facilmente reperibili e sono disponibili persino implementazioni RISC-V "open source" scritte nei linguaggi di descrizione dell'hardware. Inoltre, saranno presto disponibili delle piattaforme hardware a basso costo sulle quali si potranno eseguire programmi RISC-V. I lettori beneficeranno non solo dallo studio di queste architetture RISC-V, ma saranno anche in grado di modificarle percorrendo il processo di implementazione per capire l'impatto delle modifiche che propongono sulle prestazioni, sulla dimensione del chip e sull'energia assorbita. Questa è un'opportunità eccitante sia per l'industria dei calcolatori e sia per la didattica, e quindi quando è stato scritto questo libro, più di 40 società hanno aderito alla fondazione RISC-V. Questo elenco di sponsor comprende praticamente tutti i maggiori produttori tranne ARM e Intel, e comprende AMD, Google, Hewlett-Packard Enterprise, IBM, Microsoft, NVIDIA, Oracle e Qualcomm.

È per questi motivi che abbiamo scritto l'edizione RISC-V di questo libro, e stiamo per passare al RISC-V anche per il volume *Computer Architecture: A Quantitative Approach*.

Dato che il RISC-V offre praticamente con lo stesso insieme di istruzioni per indirizzi su 32 bit o su 64 bit, avremmo potuto passare all'insieme di istruzioni a 64 bit, ma abbiamo preferito mantenere la dimensione degli indirizzi a 32 bit. Tuttavia, il nostro editore ha intervistato i docenti che hanno adottato questo testo e hanno visto che il 75% di loro preferiva un indirizzo più ampio o erano indifferenti, per cui abbiamo allargato lo spazio di indirizzamento a 64 bit, che oggi può avere più senso di uno spazio di indirizzamento a 32 bit.

Le uniche modifiche dell'edizione RISC-V rispetto all'edizione MIPS sono quelle associate alla modifica dell'insieme delle istruzioni, che riguarda soprattutto il Capitolo 2, il Capitolo 3, la parte sulla memoria virtuale nel Capitolo 5, e i brevi esempi VMIPS del Capitolo 6. Nel Capitolo 4, siamo passati alle istruzioni RISC-V, abbiamo modificato diverse figure, e abbiamo aggiunto alcune sezioni di "Approfondimento", ma le modifiche sono state più semplici di quello che temevamo. Il Capitolo 1 e le altre Appendici sono rimaste praticamente invariate. L'estesa documentazione disponibile su web combinata con la complessità del RISC-V hanno reso difficile ottenere un'Appendice A (Gli assemblatori, i linker e il simulatore SPIM) come quella della quinta edizione MIPS. In compenso, nei Capitoli 2, 3 e 5 è riportato uno stuzzicante riassunto delle centinaia di istruzioni RISC-V che sono al di fuori delle istruzioni core RISC-V che abbiamo descritto in dettaglio nel resto del libro.

Si noti che non stiamo (ancora) dicendo che da qui in poi passeremo permanentemente al RISC-V. Ad esempio, oltre a questa nuova edizione RISC, si possono acquistare la versione ARMv8 e la versione MIPS. Una possibilità è che ci sarà richiesta per tutte e tre le edizioni di questi libro, oppure di una sola.

Decideremo quando sarà il momento. Per ora, attendiamo le vostre reazioni e i vostri feedback su questo sforzo.

Le novità di questa edizione

Nella stesura di questa edizione di *Struttura e progetto dei calcolatori* abbiamo perseguito **sei obiettivi principali**:

1. dimostrare con esempi reali quanto sia importante comprendere il funzionamento dell'hardware;
2. evidenziare i temi principali di ogni argomento inserendo a bordo pagina le icone associate, che vengono introdotte nelle pagine iniziali;
3. proporre nuovi esempi per rispecchiare i cambiamenti occorsi nel passaggio dall'era dei PC all'era post-PC;
4. distribuire il materiale relativo all'I/O su tutto il libro invece di racchiuderlo in un unico capitolo;
5. aggiornare il contenuto tecnico per rispecchiare i cambiamenti nell'industria negli anni successivi alla pubblicazione della precedente edizione;
6. spostare online le Appendici e gli altri capitoli invece di includerli in un CD, per contenere il prezzo e rendere questa edizione disponibile anche in forma elettronica.

Prima di descrivere più in dettaglio questi obiettivi, analizziamo capitolo per capitolo l'approccio adottato nella descrizione sia dell'hardware sia del software (*vedi* la **tabella** alla fine della Prefazione). I Capitoli 1, 4, 5 e 6 coprono entrambe le aree, indipendentemente dall'argomento trattato.

Il **Capitolo 1** spiega perché sia diventato importante il consumo di energia e perché questo abbia causato il passaggio dai microprocessori a processore singolo ai microprocessori multicore. In questo capitolo vengono anche introdotte le otto grandi idee sulla progettazione delle architetture degli elaboratori.

Il **Capitolo 2** è principalmente un capitolo introduttivo all'hardware, ma contiene anche una descrizione dei compilatori e dei linguaggi di programmazione a oggetti, materiale fondamentale per i lettori più interessati agli aspetti software.

Il **Capitolo 3** è dedicato all'aritmetica in virgola mobile e all'unità di elaborazione dati. I lettori possono tralasciare le parti di questo capitolo che non interessano o che contengono materiale introduttivo, ma non i paragrafi da 3.6 a 3.8. Questi descrivono una procedura che calcola il prodotto di due matrici, mostrando come il parallelismo a livello di parola consenta di migliorare le prestazioni di quattro volte.

Il **Capitolo 4** descrive i processori dotati di pipeline. I paragrafi 4.1, 4.5 e 4.10 contengono un riassunto degli argomenti trattati nel capitolo e nel paragrafo 4.12 vengono mostrate le modifiche alla procedura che esegue il prodotto di due matrici, che consentono un ulteriore aumento delle prestazioni. Anche i lettori interessati all'hardware troveranno del materiale fondamentale in questo capitolo; se le conoscenze sui circuiti logici non sono sufficienti, si può leggere l'**Appendice A** sulla progettazione dei circuiti logici, disponibile online, prima di avventurarsi nella lettura di questo capitolo.

Il **Capitolo 6**, su multicore, multiprocessori e cluster, contiene principalmente nuovo materiale e dovrebbe essere letto da tutti. Il capitolo è stato riorganizzato per rendere più naturale la successione degli argomenti e contiene una descrizione più approfondita di GPU, grandi centri di calcolo e delle interfacce hardware/software delle schede di rete, fondamentali per i cluster.

Il **primo obiettivo** di questa edizione è quello di utilizzare un esempio concreto per dimostrare quanto sia importante comprendere il funzionamento dell'hardware per ottenere buone prestazioni ed elevata efficienza energetica.

Come già accennato, inizieremo descrivendo il parallelismo a livello di parola nel Capitolo 3, che consente di ottenere un miglioramento delle prestazioni di un fattore 4 sulla moltiplicazione di due matrici. Miglioreremo ulteriormente le prestazioni nel Capitolo 4, attraverso l'espansione dei cicli, dimostrando così l'importanza del parallelismo a livello di istruzioni. Nel Capitolo 5 raddoppieremo ancora le prestazioni, ottimizzando l'utilizzo della cache mediante l'accesso a blocchi. Infine, nel Capitolo 6 mostreremo come ottenere un miglioramento di 14 volte utilizzando 16 processori e il parallelismo a livello di thread. Per ottenere tutte queste ottimizzazioni aggiungeremo solamente 24 linee di codice C alla procedura iniziale.

Il **secondo obiettivo** è aiutare i lettori a distinguere le idee fondamentali identificando all'inizio otto grandi idee nella progettazione delle architetture che vengono richiamate nel resto del volume. Abbiamo inserito le relative icone a bordo pagina ed evidenziato nel testo le parole corrispondenti per ricordare ai lettori questi otto argomenti. Questo libro contiene quasi 100 citazioni. Tutti i capitoli contengono almeno sette esempi di grandi idee, e ciascuna idea viene citata almeno cinque volte. Le prestazioni attraverso il parallelismo, la pipeline e la predizione sono tre delle idee più utilizzate, seguite da vicino dalla Legge di Moore. Il Capitolo 4, riservato al processore, è quello che contiene più esempi; ciò non deve sorprendere, poiché è il capitolo che ha probabilmente riscosso più successo presso i progettisti delle architetture digitali. La grande idea richiamata in tutti i capitoli è quella delle prestazioni attraverso il parallelismo, ben allineata all'enfasi recente sul parallelismo.

Il **terzo obiettivo** è evidenziare il passaggio generazionale nel campo dell'elaborazione dalla generazione dei PC a quella post-PC, illustrato da esempi e commenti: il Capitolo 1 analizza in dettaglio un calcolatore tablet invece di un PC e il Capitolo 6 descrive l'infrastruttura di calcolo di un cloud. In questo libro trattiamo anche l'ARM, che è l'insieme di istruzioni utilizzato nei dispositivi mobili personali dell'era post-PC, esattamente come l'insieme di istruzioni x86 ha dominato l'era dei PC e (fino a oggi) domina il mondo del cloud computing.

Il **quarto obiettivo** consiste nel distribuire il materiale relativo all'I/O su tutto il libro invece di concentrarlo in un unico capitolo, come abbiamo fatto per il parallelismo nell'edizione precedente: potete trovare il materiale sull'I/O nei paragrafi 1.4, 4.9, 5.2, 5.5, 5.11 e 6.9. Pensiamo che i lettori (e i docenti) leggeranno più volentieri il materiale sull'I/O se non è contenuto in un capitolo a sé.

Il mondo degli elaboratori è un mondo in evoluzione molto veloce e quindi, come succede sempre per le nuove edizioni, uno degli obiettivi più importanti è utilizzare contenuto tecnico aggiornato. Relativamente al **quinto obiettivo**, come esempio di architetture in questo libro abbiamo utilizzato il Cortex-A53 ARM e il Core i7 Intel, esempi tipici dei calcolatori dell'era post-PC. Altri contenuti importanti un'introduzione sulla GPU che spiega anche la sua terminologia, una trattazione più approfondita dei calcolatori dei grandi centri di calcolo che sono alla base del cloud e una descrizione dettagliata delle schede Ethernet a 10 Gigabyte.

Per il **sesto obiettivo**, contenere il prezzo e mantenere il libro di dimensioni ragionevoli e compatibile con una versione elettronica, abbiamo reso disponibili online il materiale complementare e le Appendici invece di inserirli in un CD allegato, come avveniva nelle edizioni precedenti.

Infine, abbiamo aggiornato tutti gli esercizi riportati nel libro.

Abbiamo conservato le caratteristiche delle edizioni precedenti rivelatesi più utili: abbiamo mantenuto la definizione delle parole chiave a margine del testo la prima volta che compaiono, le sezioni *Capire le prestazioni dei programmi* (dedicate alle prestazioni e a come migliorarle), le sezioni *Interfaccia hardware/software* (sui compromessi da adottare a livello di questa interfaccia), le sezioni

Quadro d'insieme (che riepilogano i concetti principali espressi nel testo) e le sezioni *Autovalutazione*, che aiutano il lettore a valutare la comprensione degli argomenti trattati (con le relative risposte esatte alla fine di ogni capitolo). Anche questa edizione contiene nell'ultima pagina una scheda tecnica riassuntiva del RISC-V. Il contenuto della scheda è stato aggiornato e costituisce un riferimento immediato per coloro che scrivono programmi nell'assembler del RISC-V.

Le risorse multimediali

online.universita.zanichelli.it/patterson5e
A questo indirizzo sono disponibili le risorse multimediali di complemento al libro. Per accedere alle risorse protette è necessario registrarsi su **my.zanichelli.it** inserendo la chiave di attivazione personale contenuta nel libro.

Libro con ebook
Chi acquista il libro può scaricare gratuitamente l'**ebook**, seguendo le istruzioni presenti nel sito. L'ebook si legge con l'applicazione *BooktabZ*, che si scarica gratis da App Store (sistemi operativi Apple) o da Google Play (sistemi operativi Android).

Considerazioni conclusive

Se leggerete il successivo paragrafo di ringraziamenti, vi renderete conto che abbiamo corretto moltissimi errori. E quando un libro passa attraverso diverse ristampe, si ha la possibilità di correggere un numero ancora maggiore di errori. Se doveste trovare altri errori, per cortesia, contattate direttamente l'editore attraverso la posta elettronica o la posta ordinaria, utilizzando gli indirizzi riportati nella pagina del copyright.

Questa edizione è la seconda dopo l'interruzione della lunga collaborazione tra Hennessy e Patterson, iniziata nel 1989. Gli impegni richiesti per dirigere una delle più importanti università del mondo hanno tolto a Hennessy il tempo necessario per lavorare alle nuove edizioni: il suo coautore, Patterson, si è sentito ancora una volta come un acrobata che si esibisce senza rete. Per questo motivo, le persone elencate nel paragrafo dei ringraziamenti e i colleghi di Berkeley hanno avuto un ruolo ancora maggiore nel dare forma al contenuto di questo libro. Ciò nonostante, uno solo è l'autore responsabile del nuovo materiale che vi apprestate a leggere.

Ringraziamenti per questa edizione

Siamo stati davvero fortunati a ricevere diversi contributi da molti lettori, redattori e ricercatori per ciascuna edizione di questo libro. Ciascuno di loro ha contribuito a rendere migliore il libro.

Siamo grati per l'assistenza di Khaled Benkrid e ai suoi colleghi di ARM Ltd. che hanno revisionato attentamente il materiale relativo agli ARM e hanno fornito interessanti suggerimenti.

Il Capitolo 6 è stato rivisto a fondo e abbiamo analizzato separatamente le idee e i contenuti; infine ho apportato le modifiche in base ai suggerimenti di ogni revisore. Vorrei ringraziare Christos Kozyrakis dell'Università di Stanford per il suggerimento di utilizzare l'interfaccia di rete dei cluster per illustrare l'interfaccia hardware/software dell'I/O e per i suggerimenti su come organizzare il capitolo; Mario Flagsilk dell'Università di Stanford, che ha fornito dettagli, diagrammi e misure delle prestazioni del NIC NetFPGA; e i seguenti ricercatori per i loro suggerimenti su come migliorare il capitolo: David Kaeli della

1 Il calcolatore: astrazioni e tecnologia

La civiltà progredisce estendendo il numero delle operazioni complesse che possiamo effettuare senza dover pensare ad esse.

Alfred North Whitehead, *An Introduction to Mathematics,* 1911

1.1 Introduzione

Benvenuti alla lettura di questo libro! È per noi un piacere mostrarvi l'eccitante mondo dei calcolatori. Questo, infatti, è tutt'altro che un mondo arido e scoraggiante, dove il progresso procede con estrema lentezza e le nuove idee finiscono per atrofizzarsi perché vengono trascurate. Tutt'altro! I calcolatori sono invece il prodotto principale di una tecnologia straordinariamente vitale, quella dell'informazione, che contribuisce per circa il 10% al prodotto interno lordo degli Stati Uniti d'America, la cui economia stessa è diventata in parte dipendente da quel rapido miglioramento della tecnologia dell'informazione promesso dalla Legge di Moore.

In questo particolare settore industriale, l'innovazione viene introdotta con una frequenza estremamente elevata. Negli ultimi 30 anni sono stati proposti diversi nuovi calcolatori la cui introduzione sembrava dovesse rivoluzionare l'intera industria degli elaboratori; invece queste rivoluzioni hanno avuto vita breve, ma solo perché altri calcolatori, più performanti, hanno sostituito rapidamente i calcolatori più lenti.

Questa corsa all'innovazione ha portato a progressi mai visti prima, già a partire dalla nascita del primo calcolatore elettronico, alla fine degli anni '40 del secolo scorso. Se l'industria dei trasporti avesse tenuto il passo di quella dei calcolatori, oggi si potrebbe andare da New York a Londra in circa un secondo spendendo solo qualche centesimo di dollaro. Fermandoci un momento a riflettere su come ciò modificherebbe la nostra società – si potrebbe

abitare a Tahiti, lavorare a San Francisco e recarsi a Mosca la sera per vedere un balletto al Bolshoi – è abbastanza facile apprezzare le opportunità che verrebbero offerte.

I calcolatori hanno dato vita alla terza rivoluzione nella nostra società, quella dell'informazione, che segue la rivoluzione agraria e quella industriale. La conseguente moltiplicazione delle potenzialità e delle capacità intellettive del genere umano ha avuto un impatto profondo sulla nostra vita quotidiana e ha cambiato il modo in cui viene generata nuova conoscenza. Esiste oggi un nuovo filone della ricerca scientifica in cui scienziati esperti di calcolo automatico collaborano con quelli che si occupano degli aspetti teorici e sperimentali al fine di esplorare nuove frontiere dell'astronomia, della biologia, della chimica, della fisica ecc.

La rivoluzione introdotta dai calcolatori è ancora in corso e ogni volta che il costo di elaborazione diminuisce di un fattore 10, i possibili utilizzi dei calcolatori si moltiplicano: applicazioni che un tempo non erano economicamente convenienti diventano improvvisamente possibili. Fino a pochi anni fa, le seguenti applicazioni erano considerate "fantascienza informatica":

- *Calcolatori nelle automobili*: fino a quando i progressi nel campo dei microprocessori non hanno prodotto, agli inizi degli anni '80, un abbattimento dei prezzi e un aumento delle prestazioni, l'idea di un controllo computerizzato delle automobili era semplicemente ridicola. Oggi gli elaboratori consentono di ridurre l'inquinamento, di migliorare l'efficienza del carburante controllando l'iniezione e la combustione e di incrementare la sicurezza. Gli elaboratori avvisano il conducente quando qualcosa è presente nell'angolo cieco durante un parcheggio o quando un veicolo si trova nella corsia adiacente quando si inizia il cambio della corsia di marcia, e controllano l'apertura degli air bag per proteggere gli occupanti della vettura in caso di incidente.
- *Telefoni cellulari*: chi avrebbe potuto immaginare che i progressi nelle architetture di elaborazione avrebbero fatto sì che più di metà degli abitanti della Terra possedesse un telefono cellulare, consentendo alle persone di comunicare tra loro praticamente da qualsiasi parte del globo?
- *Mappatura del genoma umano*: il costo dei sistemi di calcolo necessari a mappare e analizzare le sequenze del DNA umano era dell'ordine delle centinaia di milioni di dollari. È inverosimile che qualcuno potesse lanciare un simile progetto se il costo dei calcolatori fosse stato 10 o 100 volte superiore a quello attuale, come lo era 15 o 25 anni fa. Inoltre, il costo della mappatura continua a diminuire, al punto che potreste presto essere in grado di acquisire il vostro genoma, e ciò vi consentirà di personalizzare le vostre cure mediche.
- *World Wide Web*: il World Wide Web, che non esisteva quando nacque la prima edizione di questo libro, ha trasformato la nostra società. Per molte persone, il web ha sostituito le biblioteche tradizionali e i quotidiani.
- *Motori di ricerca*: con l'aumento delle dimensioni del web, e del suo valore, trovare informazioni rilevanti è diventato sempre più importante. Oggigiorno, molte persone basano sui motori di ricerca una buona parte delle loro attività, al punto che non potrebbero più farne a meno.

Augmented reality (realtà aumentata): l'arricchimento della percezione sensoriale umana, visiva in particolare, mediante informazioni pertinenti convogliate elettronicamente.

È evidente come i progressi di questa tecnologia riguardino praticamente tutti gli aspetti della nostra società. L'evoluzione degli elaboratori ha consentito la creazione di programmi meravigliosamente utili, il che spiega come mai i calcolatori siano diventati onnipresenti. Le applicazioni immaginate oggi dalla fantascienza suggeriscono le applicazioni vincenti del futuro: per esempio sono già in fase di sviluppo avanzato occhialini per l'**augmented reality** (realtà aumentata), modelli di società senza denaro contante e automobili che si guidano da sole.

Tipi di calcolatori e loro caratteristiche

Nonostante calcolatori molto diversi tra loro condividano la stessa tecnologia hardware (parr. 1.4 e 1.5), da quelli usati negli elettrodomestici più avanzati ai telefoni cellulari, fino ai supercomputer più performanti, nella maggior parte dei casi le soluzioni utilizzate non sono identiche. Infatti, queste applicazioni sono caratterizzate da requisiti di progetto differenti che implicano un diverso utilizzo dell'hardware. A grandi linee, i calcolatori vengono raggruppati in tre classi ben distinte.

I **personal computer** (**PC** – calcolatori personali) rappresentano il tipo di calcolatore più conosciuto, che molti lettori di questo libro avranno già ampiamente utilizzato. I personal computer offrono buone prestazioni a un singolo utente mantenendo il costo limitato; inoltre, vengono solitamente utilizzati per eseguire software scritto da terze parti. L'evoluzione di molte tecnologie legate ai sistemi di elaborazione è guidata proprio dai personal computer, che hanno appena 35 anni di vita!

Personal Computer (PC): un calcolatore progettato per essere utilizzato da un unico utente alla volta; solitamente comprende anche un monitor grafico, una tastiera e un mouse.

I **server** sono la forma moderna di quelli che un tempo erano calcolatori di dimensioni decisamente maggiori, e, di norma, ad essi si accede solo attraverso la rete. I server sono orientati all'elaborazione di carichi di lavoro di grosse dimensioni, rappresentati sia da singole applicazioni complesse, quali tipicamente quelle scientifiche o ingegneristiche, sia dalla gestione di tante piccole applicazioni, come nel caso di un grande server per il web. Queste applicazioni sono spesso basate su software proveniente da terze parti (per es. un database oppure un sistema di simulazione), ma sono il più delle volte modificate e adattate per svolgere una particolare funzione. I server sono realizzati con le stesse tecnologie base dei personal computer, ma offrono una maggiore potenza di calcolo, una maggiore velocità di input/output e una maggiore capacità della memoria. In generale, anche i progettisti dei server danno più importanza all'affidabilità, poiché un blocco del funzionamento di uno di questi sistemi è di solito più problematico del blocco di un PC, utilizzato da un singolo utente.

Server: un calcolatore progettato per l'esecuzione di programmi di grosse dimensioni e per servire più utenti spesso simultaneamente. Tipicamente a questi calcolatori si accede solo via rete.

I server coprono il più ampio spettro per quanto riguarda i costi e le prestazioni. I server di fascia inferiore possono essere costruiti con poco più di un comune PC senza schermo e tastiera, e costare un migliaio di dollari. Questi server di fascia bassa sono tipicamente usati per il salvataggio dei dati, per piccole applicazioni commerciali, o come semplici web server. All'estremo opposto ci sono i **supercomputer**, che attualmente sono formati da diverse decine di migliaia di processori, con memoria principale di diversi **terabyte**. Il costo di un supercomputer va dalle decine alle centinaia di milioni di dollari. I supercomputer sono tipicamente usati nel calcolo intensivo di tipo scientifico e ingegneristico, come per esempio per le previsioni meteorologiche, nelle esplorazioni petrolifere, nella determinazione delle strutture delle proteine e in altri problemi di grandi dimensioni. Nonostante i supercomputer consentano la massima capacità di calcolo, essi costituiscono solo una frazione relativamente piccola dei server esistenti e, in termini di fatturato, del mercato complessivo dei calcolatori.

Supercomputer: calcolatori con il costo più elevato e le prestazioni migliori; sono tipicamente configurati come server e possono costare decine o centinaia di milioni di dollari.

Terabyte (TB): misura di capacità pari a 1 099 511 627 776 (2^{40}) byte; gli sviluppatori di sistemi di comunicazione e di memoria di massa hanno iniziato a utilizzare questo termine per indicare la cifra di 1 000 000 000 000 (10^{12}) byte. Per evitare confusione, utilizziamo qui il termine **Tebibyte (TiB)** per indicare 2^{40} byte e il termine **Terabyte (TB)** per indicare 10^{12} byte. L'elenco completo delle unità di misura decimali e binarie e del loro valore è riportato in **Figura 1.1**.

I **calcolatori embedded** (cioè dedicati) sono i più numerosi e coprono un ampio spettro di applicazioni e prestazioni. Essi comprendono tutti i microprocessori che potete trovare nella vostra automobile, i calcolatori presenti nei televisori e la rete di processori che controlla i moderni aeroplani o le navi commerciali. I sistemi di calcolo di tipo embedded sono progettati per eseguire una singola applicazione o un insieme di applicazioni correlate tra loro; queste applicazioni sono di norma integrate con l'hardware e si presentano all'utente come un sistema monolitico. Nonostante l'enorme numero di calcolatori di tipo embedded che usano quotidianamente, molti utenti non si renderanno mai conto che stanno in realtà usando un vero e proprio computer!

Calcolatore embedded: un calcolatore posto all'interno di un altro dispositivo e usato esclusivamente per eseguire una predeterminata applicazione o un insieme di programmi.

In questo e nei capitoli successivi, troverete probabilmente molte parole nuove oppure parole che potete aver sentito ma del cui significato non siete sicuri: niente paura! È vero, ci sono molti termini speciali utilizzati per descrivere i calcolatori moderni ma, come in tutti gli ambiti scientifici, la terminologia in realtà aiuta perché consente di descrivere con precisione le funzionalità. Inoltre i progettisti di calcolatori (autori inclusi) *amano* usare gli **acronimi**, che sono facili da capire una volta che si conosce il loro significato. Per aiutare il lettore a ricordare e riconoscere i vari termini, il significato di ogni acronimo verrà riportato a lato della pagina la prima volta che l'acronimo apparirà nel testo. In breve tempo il lettore sarà in grado di utilizzare facilmente gli acronimi, e potrà impressionare gli amici con la sua capacità di usare correttamente termini quali BIOS, CPU, DIMM, DRAM, PCIe, SATA ecc.

Acronimo: una parola creata prendendo le lettere iniziali di una sequenza di parole. Per esempio: **RAM** è l'acronimo di *Random Access Memory* e **CPU** è l'acronimo di *Central Processing Unit*.

Per sottolineare ulteriormente come software e hardware possano influenzare le prestazioni di un elaboratore, troverete in diversi punti del libro sezioni speciali intitolate *Capire le prestazioni dei programmi* che riassumono gli elementi determinanti per le prestazioni dei programmi. La prima di queste sezioni è riportata qui di seguito.

Capire le prestazioni dei programmi

Le prestazioni di un programma dipendono dalla combinazione di diversi fattori: efficienza degli algoritmi implementati nel programma, efficienza del software utilizzato per creare e tradurre il programma in linguaggio macchina, ed efficienza con cui il calcolatore esegue le istruzioni, che possono comprendere anche operazioni di input/output (I/O). Questi concetti sono riassunti nella tabella che segue.

Componente hardware o software	Come la componente influenza le prestazioni	Dove viene trattato questo argomento
Algoritmi	Determina il numero di linee del codice ad alto livello e il numero di operazioni di I/O da eseguire	Altri libri!
Linguaggi di programmazione, compilatori e architetture	Determinano il numero di istruzioni in linguaggio macchina per ogni istruzione ad alto livello	Capitoli 2 e 3
Processore e sistema di memoria	Determinano quanto velocemente possono essere eseguite le istruzioni ad alto livello	Capitoli 4, 5 e 6
Sistema di I/O (hardware e sistema operativo)	Determina quanto velocemente possono essere eseguite le operazioni di I/O	Capitoli 4, 5 e 6

Per mostrare l'impatto delle idee descritte in questo libro, come accennato poco fa, mostreremo nei diversi capitoli come possano essere progressivamente migliorate le prestazioni di un programma C che moltiplica una matrice con un vettore fino a ottenere un miglioramento finale di un fattore 200! Ciascun miglioramento è basato sullo sfruttamento ottimale di un particolare componente dell'hardware del microprocessore sottostante.

- Nell'ambito del *parallelismo a livello di dati*, nel Capitolo 3, utilizzeremo le *funzioni intrinseche del C* per implementare il *parallelismo a livello di parola*, aumentando le prestazioni di un fattore 3,8.
- Nell'ambito del *parallelismo a livello di istruzioni*, nel Capitolo 4, utilizzeremo *l'espansione dei cicli per sfruttare l'esecuzione di pacchetti di istruzioni multiple e l'esecuzione hardware fuori-ordine* per aumentare le prestazioni di un ulteriore fattore 2,3.

- Nell'ambito dell'*ottimizzazione della gerarchia delle memorie*, nel Capitolo 5, utilizzeremo il *blocco della cache* per aumentare le prestazioni su matrici di grandi dimensioni, con un miglioramento di un ulteriore fattore da 2 a 2,5.
- Nell'ambito del *parallelismo a livello di thread*, nel Capitolo 6, utilizzeremo *cicli for paralleli implementati in OpenMP per sfruttare l'hardware multicore*, aumentando le prestazioni di un altro fattore da 4 a 14.

Autovalutazione

Le sezioni denominate *Autovalutazione* sono pensate per aiutare il lettore a valutare se abbia compreso i principali concetti introdotti nel capitolo e per mostrarne a fondo le implicazioni. Alcune domande di queste sezioni sono molto semplici mentre altre hanno lo scopo di aprire una discussione con altre persone. Potete trovare le risposte alle singole domande alla fine del capitolo. Le domande di autovalutazione sono inserite al termine dei paragrafi, in modo da poterle evitare se siete sicuri di aver appreso quanto avete letto.

1. Il numero di calcolatori embedded venduti ogni anno supera largamente quello dei PC e persino quello dei calcolatori post-PC. Siete in grado di confermare o confutare questa affermazione basandovi sulla vostra esperienza? Provate a contare il numero di processori embedded presenti nella vostra abitazione e confrontatelo con il numero di calcolatori tradizionali. Qual è il risultato di questo confronto?
2. Come accennato precedentemente, sia il software sia l'hardware influenzano le prestazioni di un programma. Provate a pensare ad alcuni esempi pratici nei quali ognuno dei seguenti elementi può rappresentare un "collo di bottiglia" per le prestazioni:
   - algoritmo implementato;
   - linguaggio di programmazione o compilatore;
   - sistema operativo;
   - processore;
   - sistema di I/O e periferiche.

1.2 Otto grandi idee sull'architettura dei calcolatori

Presentiamo ora otto grandi idee dei progettisti dei calcolatori degli ultimi 60 anni. Queste idee sono state così innovative da durare a lungo dopo la loro implementazione nelle prime architetture: i progettisti più giovani hanno ripreso queste idee nella progettazione delle architetture più recenti. Queste idee sono dei principi ricorrenti che compariranno spesso in questo capitolo e nei capitoli successivi; per identificarle introduciamo qui i simboli che le rappresentano e i termini ad esse associati. Utilizzeremo questi simboli per individuare i quasi 100 paragrafi di questo libro che descrivono le caratteristiche basate su queste idee.

Progettare tenendo conto della Legge di Moore

LEGGE DI MOORE

Una delle costanti dei progettisti di calcolatori è la rapida evoluzione, descritta principalmente dalla **Legge di Moore**. Essa stabilisce che le risorse messe a disposizione dai circuiti integrati vengano duplicate ogni 18-24 mesi. La Legge di Moore deriva da una predizione sull'aumento della capacità dei circuiti integrati fatta nel 1965 da Gordon Moore, uno dei fondatori di Intel. Dato che la progettazione di un calcolatore può richiedere anni, le risorse messe a disposizione da un chip possono facilmente raddoppiare o quadruplicare prima della realizzazione finale del calcolatore. Come nel tiro al piattello, i progettisti dei

calcolatori devono indovinare dove sarà la tecnologia quando sarà terminata la fase di progettazione e non progettare in base alla tecnologia attuale. Utilizzeremo una freccia verso l'alto e verso destra come simbolo che rappresenta la Legge di Moore e l'evoluzione rapida che essa rappresenta.

Utilizzo delle astrazioni per semplificare il progetto


ASTRAZIONE

Sia i progettisti dei calcolatori sia i programmatori hanno dovuto inventare delle tecniche che li rendessero più produttivi, per evitare che il tempo riservato alla progettazione crescesse enormemente con il crescere secondo la Legge di Moore delle risorse rese disponibili. Una di queste tecniche è l'**astrazione**, utilizzata per rappresentare il progetto sia hardware sia software a diversi livelli di definizione: i dettagli vengono nascosti ai livelli più bassi per offrire un modello più semplice ai livelli più alti. Utilizzeremo un disegno astratto come simbolo per questa seconda grande idea.

Rendere veloci le situazioni più comuni


SITUAZIONI COMUNI

Rendere veloci le situazioni più comuni tende a fare aumentare le prestazioni più dell'ottimizzazione delle funzionalità utilizzate raramente. Ironicamente, le situazioni più comuni sono spesso più semplici di quelle rare e quindi di solito sono più semplici da migliorare. Questo consiglio dettato dal buon senso prevede che sappiate quali sono le situazioni più comuni, cosa che è possibile solo attraverso un'attenta sperimentazione e analisi (par. 1.6). Utilizzeremo il simbolo di un'auto sportiva per rappresentare quest'idea, dato che i viaggi più frequenti prevedono uno o due passeggeri, ed è sicuramente più facile realizzare una macchina sportiva veloce che un minivan veloce!

Prestazioni attraverso il parallelismo


PARALLELISMO

Sin dagli albori, i progettisti hanno realizzato calcolatori che ottengono prestazioni più elevate eseguendo le operazioni in parallelo. Vedremo molti esempi di parallelismo in questo libro. Utilizzeremo il simbolo di un aereo a reazione con più motori per rappresentare le prestazioni attraverso il **parallelismo**.

Prestazioni attraverso la pipeline


PIPELINE

Una particolare forma di parallelismo è così diffusa nelle architetture da meritarsi un nome proprio: **pipeline**. Per esempio, una squadra di uomini, prima di chiamare i vigili del fuoco, può cercare di spegnere il fuoco con i secchi, come avviene tipicamente nei film western quando il cattivo appicca il fuoco: gli abitanti del paese formano una catena umana per trasportare l'acqua dalla sorgente al fuoco. Infatti, in questo modo si riesce a trasportare l'acqua dalla sorgente al fuoco più velocemente che correndo avanti e indietro. Il simbolo della pipeline è una sequenza di condotti, dove ciascun condotto rappresenta uno stadio della pipeline.

Prestazioni attraverso la predizione


PREDIZIONE

Seguendo il detto secondo cui "è meglio chiedere perdono che chiedere permesso", un'altra grande idea è la **predizione**. In alcuni casi, è in genere più veloce tirare a indovinare e iniziare a lavorare di conseguenza, che aspettare di sapere con certezza. Questo è vero quando il meccanismo per recuperare una predizione sbagliata non è troppo costoso e la predizione è sufficientemente accurata. Utilizzeremo la sfera di cristallo come simbolo della predizione.

Gerarchia delle memorie


GERARCHIA

I programmatori vogliono che la memoria sia di grandi dimensioni, veloce e poco costosa, dato che la sua velocità spesso determina le prestazioni, la sua capacità limita la dimensione dei problemi che possono essere risolti e il suo costo rappresenta la voce di costo maggiore di un'architettura. I progettisti hanno scoperto che possono soddisfare queste esigenze contrapposte con la **gerarchia delle memorie**, nella quale la memoria più veloce, piccola e con il maggior costo per bit si trova in cima alla gerarchia e la memoria più lenta, più grande e con il minore costo per bit alla base. Come vedremo nel Capitolo 5, le memorie cache forniscono al programmatore l'illusione che la memoria principale sia veloce quasi quanto la memoria in cima alla gerarchia e sia economica e grande quasi quanto quella alla base della gerarchia. Utilizzeremo come simbolo della gerarchia delle memorie un triangolo a strati. La forma indica la velocità, il costo e la dimensione: più vicino è lo strato di memoria alla cima, maggiore sarà la sua velocità e il suo costo; più larga è la base, maggiore sarà la capacità della memoria.

Affidabilità e ridondanza


AFFIDABILITÀ

I calcolatori non devono solo essere veloci, devono anche essere affidabili. Dato che tutti i dispositivi elettronici sono soggetti a guasti, per renderli **affidabili** dobbiamo introdurre dei componenti ridondanti che possano essere attivati quando si verifica un guasto *e* per aiutare a identificare i guasti stessi. Utilizzeremo come simbolo di affidabilità e ridondanza un bilico, dato che la doppia coppia di ruote degli assi posteriori consente al camion di continuare a viaggiare anche quando una ruota si sgonfia (anche se immaginiamo che il camionista si recherà subito presso un gommista per riparare la ruota, restaurando così la ridondanza!).

1.3 Che cosa c'è dietro un programma

A Parigi la gente semplicemente si stupiva quando le parlavo in francese; non riuscii mai a fare comprendere a quegli idioti la loro lingua.

Mark Twain, *Gli innocenti all'estero,* 1869.


ASTRAZIONE

Una tipica applicazione, come un programma per la scrittura di testi o un grande database, è costituita da milioni di linee di codice e si serve di sofisticate librerie software che implementano funzioni complesse di supporto all'applicazione stessa. Come vedremo più avanti, il calcolatore può solo eseguire istruzioni di basso livello estremamente semplici. Passare da un'applicazione complessa (scritta in un linguaggio vicino a quello dell'essere umano) alle semplici istruzioni che possono essere comprese dal calcolatore è un processo che coinvolge diversi strati di software che interpretano e traducono le operazioni definite ad alto livello nelle semplici istruzioni comprensibili al calcolatore. Questo è un esempio di **astrazione**.

Questi strati di software sono organizzati principalmente in maniera gerarchica come mostrato in **Figura 1.3**. Nel cerchio più esterno compaiono le applicazioni, mentre i diversi componenti del **software di sistema** sono posizionati nel cerchio intermedio tra l'hardware e le applicazioni software.

Software di sistema: software che fornisce servizi di comune utilità. Comprende i sistemi operativi, i compilatori e gli assemblatori.

Il software di sistema ha diversi componenti, ma due sono quelli essenziali per tutti i calcolatori moderni: il sistema operativo e il compilatore.

Sistema operativo: programma di supervisione che gestisce le risorse di un calcolatore a vantaggio dei programmi eseguiti su quel calcolatore.

Il **sistema operativo** permette di interfacciare i programmi utente con l'hardware del calcolatore, fornendo un gran numero di servizi e funzioni di supervisione. Alcune tra le funzioni più importanti del sistema operativo sono:

- gestire le operazioni base di input/output;
- allocare spazio nella memoria principale e nei dispositivi di memoria di massa;
- consentire a più applicazioni di utilizzare simultaneamente e in modo sicuro lo stesso calcolatore.

Linux, iOS e Windows sono tre esempi di sistemi operativi oggi largamente utilizzati.

I **compilatori** eseguono un'altra funzione vitale: la traduzione di un programma scritto in un linguaggio ad alto livello, come per esempio C, C++, Java o Visual Basic, in istruzioni eseguibili dall'hardware. Data la complessità dei moderni linguaggi di programmazione e la semplicità delle istruzioni che possono essere eseguite dall'hardware, la traduzione del codice scritto ad alto livello in istruzioni hardware è una funzione complessa. Introduciamo ora brevemente il processo di traduzione, che sarà trattato in modo più approfondito nel Capitolo 2.

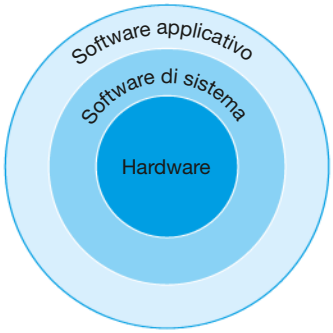

Figura 1.3 Schema semplificato della relazione gerarchica tra hardware e software rappresentati come cerchi concentrici: l'hardware è rappresentato dal cerchio più interno e il software applicativo da quello più esterno. In applicazioni complesse ci sono spesso più livelli di applicazioni software. Per esempio, un database può essere eseguito al di sopra del software di sistema, che ha lanciato un'applicazione, a sua volta eseguita sopra il database.

Da un linguaggio ad alto livello al linguaggio dell'hardware

Per parlare con una macchina elettronica è necessario inviare segnali elettrici; i segnali che un calcolatore può comprendere più facilmente sono *on* e *off* (acceso e spento), quindi l'alfabeto della macchina consta di due soli simboli. Proprio come le 26 lettere che costituiscono l'alfabeto inglese non pongono alcun limite a quante parole si possano scrivere, il fatto che l'alfabeto dei calcolatori sia costituito soltanto da due simboli non pone un limite a quello che può fare un calcolatore. Questi due simboli sono rappresentati dai numeri 0 e 1; il linguaggio macchina può quindi essere visto come una composizione di numeri in base 2, detti anche *numeri binari*, dove ogni "lettera" è una **cifra binaria** (*binary digit*), detta **bit**. I calcolatori obbediscono ai nostri comandi, chiamati istruzioni. Un'**istruzione** è semplicemente una stringa, o insieme, di bit comprensibile dal calcolatore e, perciò, può essere vista come un numero. Per esempio, la sequenza di bit:

```
1001010100101110
```

può ordinare a un particolare calcolatore di sommare due numeri. Il motivo per cui si utilizzano i numeri per rappresentare sia le istruzioni sia i dati verrà spiegato nel Capitolo 2. Non vogliamo anticiparne il contenuto, qui osserviamo solamente che ciò costituisce una delle idee fondamentali attorno alle quali sono costruiti i calcolatori.

I primi programmatori comunicavano con il calcolatore direttamente tramite numeri binari, ma era così macchinoso che in breve tempo furono inventate nuove notazioni assai più vicine al modo di pensare dell'essere umano. All'inizio tali notazioni venivano tradotte manualmente nel codice binario corrispondente, ma questo processo era ancora estenuante.

I pionieri dell'informatica allora inventarono del software in grado di tradurre una notazione simbolica in codice binario, usando il calcolatore stesso per programmare il calcolatore. Il primo di questi programmi fu chiamato **assembler** (**assemblatore**): esso traduceva la versione simbolica delle istruzioni nella corrispondente forma binaria. Per esempio, un programmatore potrebbe scrivere:

```
add A, B
```

e l'assemblatore tradurrebbe questa notazione in:

```
1001010100101110
```

Questa istruzione dice al calcolatore di sommare i due numeri A e B. Il nome coniato per questo tipo di linguaggio simbolico, usato ancora oggi, è **linguaggio assembler** (*assembly language*), mentre il linguaggio binario, compreso dal calcolatore, prende il nome di **linguaggio macchina**.

Sebbene il linguaggio assembler costituisca un notevole progresso, è ancora lontano dal costituire una notazione che uno scienziato vorrebbe utilizzare per

Compilatore: programma che traduce le istruzioni scritte in linguaggio ad alto livello in istruzioni assembler.

Cifra binaria: detta anche **bit**, è uno dei due numeri previsti dalla base 2 (0 o 1). È il componente elementare dell'informazione digitale.

Istruzione: un comando che l'hardware del computer comprende ed esegue.

Assembler (assemblatore): un programma che converte una versione simbolica delle istruzioni in una versione binaria comprensibile alla macchina.

Linguaggio assembler: rappresentazione simbolica delle istruzioni in linguaggio macchina.

Linguaggio macchina: rappresentazione binaria delle istruzioni comprensibile dall'hardware di un calcolatore.

Programma in linguaggio ad alto livello (in C)

```
scambia(size_t v[], size_t k)
{
   size_t temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

Compilatore

Programma in linguaggio assembler (per RISC-V)

```
scambia:
        slli x6, x11, 3
        add  x6, x10, x6
        ld   x5, 0(x6)
        ld   x7, 8(x6)
        sd   x7, 0(x6)
        sd   x5, 8(x6)
        jalr x0, 0(x1)
```

Assemblatore

Programma in linguaggio macchina binario (per RISC-V)

```
00000000001101011001001100010011
00000000011001010000001100110011
00000000000000011001001010000011
00000000100000011001001110000011
00000000011100110011000000100011
00000000010100110011010000100011
00000000000000000100000001100111
```

Figura 1.4 Programma in linguaggio C, compilato in linguaggio assembler e quindi tradotto (assemblato) in linguaggio macchina. Anche se la traduzione da linguaggio ad alto livello a linguaggio macchina viene qui suddivisa in due fasi, alcuni compilatori eliminano lo stadio intermedio producendo direttamente il codice in linguaggio macchina binario. Questi linguaggi e questo programma verranno esaminati più nel dettaglio nel Capitolo 2.

Linguaggio di programmazione ad alto livello: un linguaggio portabile come C, C++, Java o Visual Basic, composto da parole ed espressioni algebriche che possono essere tradotte da un compilatore in linguaggio assembler.

simulare il flusso di un fluido o che un amministratore potrebbe impiegare per la gestione dei propri registri contabili. Il linguaggio assembler richiede la scrittura di una linea per ogni istruzione che il calcolatore eseguirà, forzando così il programmatore a pensare come la macchina.

Avere compreso che si poteva scrivere un programma per tradurre in linguaggio macchina istruzioni scritte in un linguaggio più potente è stato uno dei passi più importanti compiuti all'inizio dell'era informatica. I programmatori di oggi devono non solo la loro produttività ma anche la loro salute mentale alla creazione di **linguaggi di programmazione ad alto livello** e di compilatori che traducono i programmi scritti in tali linguaggi nelle istruzioni del linguaggio macchina. La **Figura 1.4** mostra la relazione tra i programmi scritti nei tre diversi linguaggi. Questo è un altro esempio della potenza dell'**astrazione**.

ASTRAZIONE

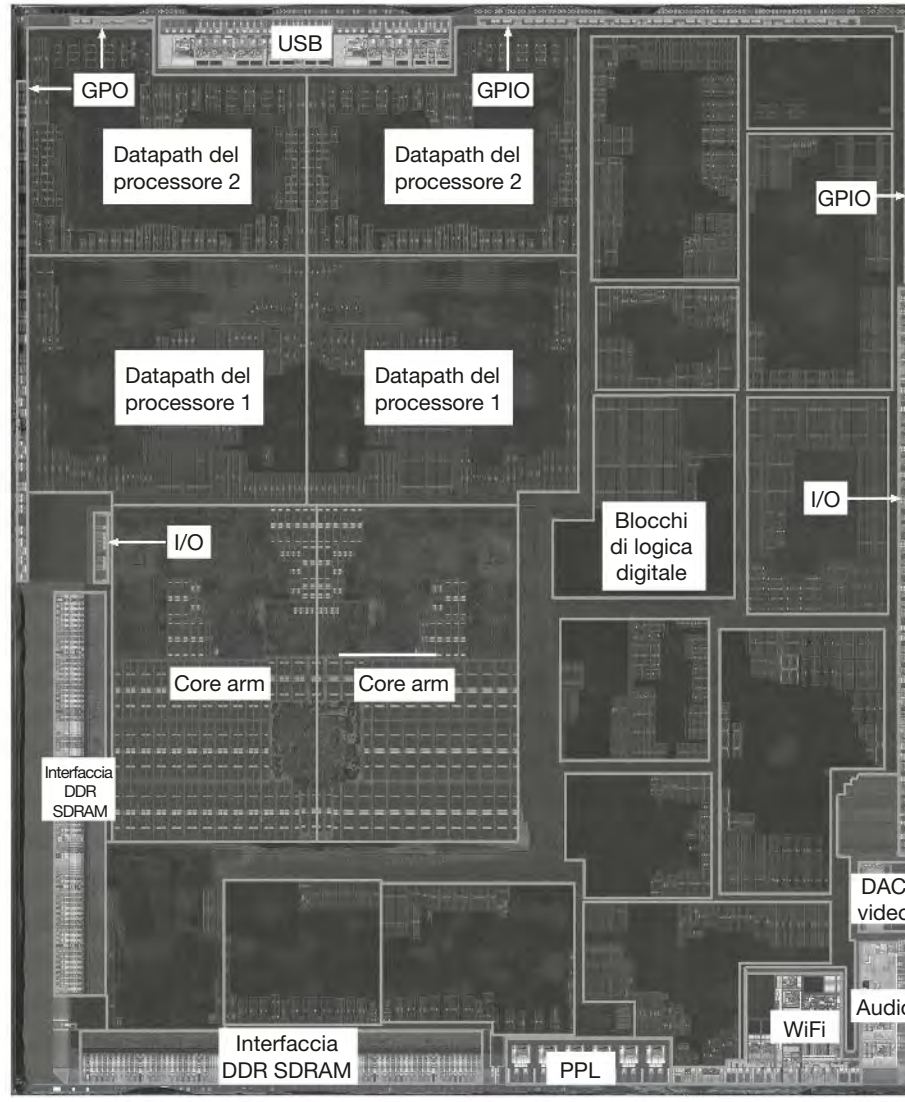

Figura 1.9 Circuito integrato del processore contenuto nel chip dell'A5. Il chip ha dimensioni di 12,1 × 10,1 mm ed è stato inizialmente prodotto con la tecnologia a 45 nm (par. 1.5). Contiene due processori, o core, ARM che sono alloggiati in mezzo a sinistra del chip e una GPU (unità di elaborazione grafica) PowerVR contenente quattro unità di elaborazione dati che potete vedere in alto a sinistra. A sinistra in basso potete vedere le interfacce verso la memoria principale (DRAM) (*Fonte*: Chipworks, www.chipworks.com).

Architettura dell'insieme di istruzioni (ISA): detta anche semplicemente **architettura**. Interfaccia astratta tra l'hardware e il livello più basso del software di un calcolatore. Comprende tutte le informazioni necessarie per scrivere un programma in linguaggio macchina funzionante in modo corretto, comprese le istruzioni, i registri, gli accessi a memoria, l'I/O ecc.

ASTRAZIONE

Come abbiamo appena visto, una delle grandi idee per migliorare i calcolatori è l'astrazione e una delle **astrazioni** più importanti è l'interfaccia tra l'hardware e il software di più basso livello. Per la sua importanza, è stato associato un nome specifico a questa interfaccia: **architettura dell'insieme di istruzioni** del calcolatore (**ISA**, *Instruction Set Architecture*), o più semplicemente **architettura** del calcolatore. L'architettura dell'insieme di istruzioni comprende tutto ciò che i programmatori devono sapere per scrivere un programma in linguaggio macchina correttamente funzionante e comprende quindi le istruzioni, i dispositivi di I/O ecc. Tipicamente, un sistema operativo incapsula i dettagli sull'I/O, sull'allocazione della memoria e su altre informazioni di basso livello, evitando quindi al programmatore di dover gestire questi dettagli. La combinazione dell'insieme di base delle istruzioni e dell'interfaccia del sistema operativo fornita ai programmatori per scrivere le applicazioni

è chiamata **interfaccia binaria delle applicazioni** (**ABI**, *Application Binary Interface*).

Un'architettura dell'insieme di istruzioni permette ai progettisti di descrivere le funzionalità di un calcolatore in maniera completamente indipendente dall'hardware che le implementa. Per esempio, si può parlare delle funzioni di un orologio digitale (misurare il tempo, visualizzare l'ora, programmare la sveglia) indipendentemente dalla tecnologia hardware con cui l'orologio viene realizzato (schermo a cristalli liquidi, schermo a LED, tasti in plastica). Tutti i progettisti di calcolatori distinguono fra architettura e **implementazione** di un'architettura allo stesso modo: un'implementazione di un'architettura è un hardware che realizza la descrizione astratta dell'architettura. Questi concetti ci portano verso un altro *Quadro d'insieme*.

QUADRO D'INSIEME

Sia l'hardware sia il software consistono in diversi livelli gerarchici basati su diversi livelli di astrazione, ognuno dei quali nasconde alcuni dettagli al livello superiore. Questo principio di astrazione consente sia ai progettisti hardware sia ai progettisti software di affrontare la complessità della progettazione dei calcolatori. Un'interfaccia chiave tra i due livelli di astrazione è l'*architettura dell'insieme di istruzioni* (ISA, *Instruction Set Architecture*), ossia l'interfaccia tra l'hardware e il software di basso livello. Quest'interfaccia astratta permette diverse *implementazioni* hardware, caratterizzate da diversi costi e prestazioni, per eseguire lo stesso software. ■

Un posto sicuro per i dati

Finora abbiamo visto come inserire dati, eseguire calcoli su di essi e visualizzare il risultato. Tuttavia, se dovesse interrompersi l'alimentazione del calcolatore, tutto andrebbe perso perché la memoria interna del calcolatore è una **memoria volatile**. Questo significa che quando viene a mancare l'alimentazione, la memoria dimentica tutto. Al contrario, quando si spegne il lettore DVD, il disco DVD non dimentica mai il film che vi è stato registrato sopra; questo è un esempio di **memoria non volatile**.

Per distinguere tra la memoria utilizzata per memorizzare i programmi mentre questi vengono eseguiti e la memoria non volatile utilizzata per salvare i programmi tra un'esecuzione e l'altra, si utilizza il termine **memoria principale** (o **memoria primaria**) per la prima e **memoria di massa** (o **memoria secondaria**) per la seconda. La memoria di massa costituisce il secondo strato della **gerarchia delle memorie**. Le DRAM hanno dominato il settore delle memorie principali sin dal 1975, mentre i **dischi magnetici** (o **disco rigido** – **hard disk**) hanno dominato quello delle memorie di massa ben da prima. Per la loro dimensione e la loro forma, i dispositivi mobili utilizzano le **memorie flash**, memorie non volatili a semiconduttore, al posto dei dischi rigidi. La Figura 1.8 mostra un chip che contiene la memoria flash di un iPad2. Sebbene più lente delle DRAM, le memorie flash costano molto meno e possiedono anch'esse la caratteristica della non volatilità. Anche se hanno un costo per bit maggiore dei dischi, le memorie flash sono più compatte, sono disponibili con capacità inferiori, sono più robuste e consumano meno corrente dei dischi. A differenza dei dischi e delle DRAM, il contenuto di un bit di una memoria flash tende a svanire dopo essere stato riscritto da 100 000 a 1 000 000 di volte. Per evitare ciò, il file system deve tenere traccia del numero di scritture e adottare una strategia

Interfaccia binaria delle applicazioni (ABI): è il sottoinsieme di istruzioni dedicate all'utente più l'interfaccia del sistema operativo fornita al programmatore per scrivere le applicazioni. Definisce uno standard per la portabilità del codice binario tra calcolatori differenti.

Implementazione: hardware che rispetta le specifiche imposte dalla descrizione astratta di un'architettura.

Memoria volatile: memoria in grado di mantenere i dati solamente se è alimentata. Un tipico esempio di memoria volatile sono le DRAM.

Memoria non volatile: memoria che conserva i dati anche quando viene a mancare l'alimentazione; viene utilizzata per conservare i dati fra un'esecuzione e l'altra. I dischi DVD costituiscono una memoria non volatile.

Memoria principale: detta anche **memoria primaria**, viene utilizzata per contenere i programmi durante la loro esecuzione. Nei calcolatori moderni è tipicamente costituita da DRAM.

Memoria di massa: detta anche **memoria secondaria**, è una memoria non volatile utilizzata per conservare i programmi e i dati fra un'esecuzione e l'altra. È tipicamente una memoria flash nei PMD mentre è costituita da dischi magnetici nei server.

Disco magnetico (o disco rigido – hard disk): memoria secondaria non volatile costituita da piatti rotanti ricoperti da materiale magnetico in grado di memorizzare informazioni binarie. Dato che i dischi ruotano meccanicamente, il tempo di accesso è compreso tra i 5 e i 20 millisecondi. Il costo per Gigabyte era compreso tra 0,05 e 0,10 dollari americani nel 2012.

Memoria flash: una memoria non volatile a semiconduttori. È più lenta e meno costosa delle memorie DRAM, ma più costosa e più veloce dei dischi rigidi. Il tempo di accesso è compreso tra i 5 e i 50 microsecondi e il costo per Gigabyte era compreso tra 0,75 e 1 dollaro americano nel 2012.

Autovalutazione

1. Un'applicazione che utilizza il cloud e un PMD è limitata dalle prestazioni della rete. Stabilire se le seguenti modifiche migliorino il solo throughput, il throughput e il tempo di esecuzione, oppure nessuno dei due.
   a. Un canale di rete supplementare viene inserito tra il PMD e il cloud, aumentando così il throughput totale della rete e riducendo i ritardi di accesso alla rete (dato che sono ora disponibili due canali di rete).
   b. Il software di rete viene migliorato, riducendo così i ritardi di comunicazione sulla rete; non viene invece migliorato il throughput.
   c. Viene aggiunta altra memoria al calcolatore.
2. Le prestazioni del calcolatore C sono 4 volte migliori di quelle del calcolatore B, che esegue una certa applicazione in 28 secondi. Quanto impiega il calcolatore C per eseguire quell'applicazione?

Prestazioni della CPU

Gli utenti e i progettisti utilizzano di solito metriche di tipo diverso per valutare le prestazioni. Se si riuscissero a mettere in relazione le diverse metriche, sarebbe possibile determinare l'effetto di una modifica dell'architettura sulle prestazioni che vengono percepite dai diversi utenti. Dal momento che ora ci stiamo occupando delle prestazioni della CPU, possiamo affermare che la misura fondamentale è il tempo di esecuzione della CPU (o tempo di CPU). Le metriche più elementari (numero di cicli di clock e periodo del clock) sono legate al tempo di CPU tramite una semplice equazione:

$$\begin{array}{l}\text{Tempo di CPU relativo}\\ \text{a un programma}\end{array} = \begin{array}{l}\text{Cicli di clock della CPU}\\ \text{relativi al programma}\end{array} \times \text{Periodo di clock}$$

Poiché la frequenza di clock è il reciproco del suo periodo, è possibile utilizzare in alternativa la seguente equazione:

$$\begin{array}{l}\text{Tempo di CPU relativo}\\ \text{a un programma}\end{array} = \frac{\text{Cicli di clock della CPU relativi a un programma}}{\text{Frequenza di clock}}$$

Quest'ultima equazione chiarisce che un progettista hardware può migliorare le prestazioni riducendo il numero di cicli di clock necessari per eseguire un programma oppure la durata del ciclo di clock stesso. Come vedremo nei prossimi capitoli, i progettisti spesso devono trovare un compromesso tra il numero di cicli di clock e la durata del singolo ciclo. Molte delle tecniche che riducono il numero di cicli di clock, infatti, tendono ad aumentare la durata del periodo del clock.

Migliorare le prestazioni

ESEMPIO

Il nostro programma preferito viene eseguito in 10 secondi dal calcolatore A, che è dotato di un clock a 2 GHz. Stiamo cercando di aiutare un progettista a costruire un calcolatore B in grado di eseguire lo stesso programma in 6 secondi. Il progettista ha concluso che è possibile aumentare in modo significativo la frequenza di clock; questa modifica

(continua)

(continua)

avrà però un'influenza su tutto il progetto della CPU, facendo sì che il calcolatore B richieda un numero di cicli di clock maggiore di un fattore 1,2 rispetto al calcolatore A per eseguire il programma. Dovendo dare un consiglio al progettista, quale sarà la frequenza di clock che permette di raggiungere l'obiettivo?

SOLUZIONE

Innanzitutto determiniamo il numero di cicli di clock necessari all'esecuzione del programma su A:

$$\text{Tempo di CPU}_A = \frac{\text{Cicli di clock CPU}_A}{\text{Frequenza di clock}_A}$$

$$10 \text{ secondi} = \frac{\text{Cicli di clock CPU}_A}{2 \times 10^9 \dfrac{\text{cicli}}{\text{secondo}}}$$

$$\text{Cicli di clock CPU}_A = 10 \text{ secondi} \times 2 \times 10^9 \frac{\text{cicli}}{\text{secondo}} = 20 \times 10^9 \text{ cicli}$$

Il tempo di CPU per B si può calcolare tramite la seguente equazione:

$$\text{Tempo di CPU}_B = \frac{1{,}2 \times \text{Cicli di clock CPU}_A}{\text{Frequenza di clock}_B}$$

$$6 \text{ secondi} = \frac{1{,}2 \times 20 \times 10^9 \text{cicli}}{\text{Frequenza di clock}_B}$$

$$\text{Frequenza di clock}_B = \frac{1{,}2 \times 20 \times 10^9 \text{cicli}}{6 \text{ secondi}} = \frac{0{,}2 \times 20 \times 10^9 \text{cicli}}{\text{secondi}} = \frac{4 \times 10^9 \text{cicli}}{\text{secondi}} = 4 \text{ GHz}$$

Quindi il calcolatore B dovrà lavorare a una frequenza di clock doppia rispetto ad A per eseguire il programma in 6 secondi.

Misura delle prestazioni associate alle istruzioni

Le equazioni precedenti non contenevano alcun riferimento al numero di istruzioni presenti nel programma. Tuttavia, dato che il compilatore genera delle istruzioni da eseguire e il calcolatore esegue le singole istruzioni per eseguire il programma, il tempo di esecuzione dovrà necessariamente dipendere dal numero delle istruzioni che compongono il programma. È possibile esprimere il tempo di esecuzione totale come il risultato del prodotto del numero di istruzioni da eseguire per il tempo medio di esecuzione di ciascuna istruzione. Di conseguenza, il numero di cicli di clock necessari per l'esecuzione di un programma si può scrivere come:

$$\text{Cicli di clock della CPU} = \text{Numero di istruzioni del programma} \times \text{Numero medio di cicli di clock per istruzione}$$

Cicli di clock per istruzione (CPI): numero medio di cicli di clock per istruzione di un programma o di un frammento di programma.

Il termine **cicli di clock per istruzione**, calcolato come media del numero di cicli di clock che le diverse istruzioni richiedono per essere completate, è spesso abbreviato con la sigla **CPI**. Dato che istruzioni diverse possono richiedere un tempo di esecuzione differente in funzione del compito che svolgono, il CPI assume il valore medio calcolato su tutte le istruzioni dal programma. Il CPI è una quantità utile a confrontare due calcolatori diversi che condividono la stessa architettura dell'insieme di istruzioni, dal momento che il numero di istruzioni contenute in un programma sarà, in questo caso, lo stesso.

Come utilizzare l'equazione delle prestazioni

ESEMPIO

Siano date due implementazioni diverse della stessa architettura dell'insieme di istruzioni. Il calcolatore A ha un ciclo di clock di 250 ps e un CPI pari a 2,0 per un determinato programma; il calcolatore B, invece, ha un ciclo di clock pari a 500 ps e un CPI pari a 1,2 misurato sullo stesso programma. Quale dei due calcolatori è più veloce nell'esecuzione del programma, e di quanto?

SOLUZIONE

Sappiamo che i due calcolatori eseguono lo stesso numero di istruzioni elementari quando eseguono il programma; chiamiamo I il numero di istruzioni. Per prima cosa occorre calcolare il numero di cicli di clock richiesti per l'esecuzione da parte di ciascun calcolatore:

$$\text{Cicli di clock della CPU}_A = I \times 2{,}0$$

$$\text{Cicli di clock della CPU}_B = I \times 1{,}2$$

Si può ora calcolare il tempo di CPU per ciascun calcolatore:

$$\begin{aligned}\text{Tempo di CPU}_A &= \text{Cicli di clock della CPU}_A \times \text{Periodo di clock} \\ &= I \times 2{,}0 \times 250 \text{ ps} = 500 \times I \text{ ps}\end{aligned}$$

Analogamente, per B si ha:

$$\text{Tempo di CPU}_B = I \times 1{,}2 \times 500 \text{ ps} = 600 \times I \text{ ps}$$

È chiaro che il calcolatore A è più veloce di B, di un fattore dato dal rapporto tra i tempi di esecuzione:

$$\frac{\text{Prestazioni della CPU}_A}{\text{Prestazioni della CPU}_B} = \frac{\text{Tempo di esecuzione}_B}{\text{Tempo di esecuzione}_A} = \frac{600 \times I \text{ ps}}{500 \times I \text{ ps}} = 1{,}2$$

Si può dunque concludere che il calcolatore A è 1,2 volte più veloce di B nell'esecuzione di questo programma.

Equazione classica di misura delle prestazioni

Possiamo quindi scrivere questa equazione fondamentale in funzione del **numero di istruzioni** (eseguite da un programma), del CPI e del periodo di clock:

Numero di istruzioni: il numero delle istruzioni eseguite da un programma.

$$\text{Tempo di CPU} = \text{Numero di istruzioni} \times \text{CPI} \times \text{Periodo di clock}$$

oppure, poiché la frequenza di clock è l'inverso del periodo di clock:

$$\text{Tempo di CPU} = \frac{\text{Numero di istruzioni} \times \text{CPI}}{\text{Frequenza di clock}}$$

Le formule precedenti sono particolarmente utili in quanto evidenziano i tre fattori principali che influenzano le prestazioni. Utilizzeremo queste formule per confrontare tra loro due architetture o per valutare un'alternativa progettuale conoscendone l'impatto sui tre parametri.

Confronto di frammenti di codice

ESEMPIO

Un progettista di compilatori deve decidere quale tra due sequenze di codice implementare per un certo calcolatore. I progettisti dell'hardware gli hanno fornito queste informazioni:

	CPI per ciascun tipo di istruzione		
	A	B	C
CPI	1	2	3

Per una certa istruzione in linguaggio ad alto livello, il progettista sta considerando due sequenze di codice in linguaggio macchina che richiedono un numero diverso di istruzioni dei tre tipi:

Sequenza di istruzioni	Numero di istruzioni in linguaggio macchina per ciascun tipo di istruzione		
	A	B	C
Sequenza 1	2	1	2
Sequenza 2	4	1	1

Quale sequenza richiede l'esecuzione di un maggior numero di istruzioni? Quale verrà eseguita più velocemente? Qual è il CPI delle due sequenze?

SOLUZIONE

Nella sequenza 1 vengono eseguite: 2 + 1 + 2 = 5 istruzioni, mentre la sequenza 2 richiede 4 + 1 + 1 = 6 istruzioni; perciò la sequenza 1 richiede un numero minore di istruzioni.

Per calcolare il numero totale di cicli di clock per ciascuna sequenza, possiamo utilizzare l'equazione che misura il numero di cicli di clock della CPU in funzione del numero di istruzioni e del CPI:

$$\text{Cicli di clock della CPU} = \sum_{i=1}^{n} \left(\text{CPI}_i \times \text{C}_i\right)$$

Da questa equazione possiamo ricavare il numero totale dei cicli di clock:

$\text{Cicli di clock della CPU}_1 = (2 \times 1) + (1 \times 2) + (2 \times 3) = 2 + 2 + 6 = 10$ cicli
$\text{Cicli di clock della CPU}_2 = (4 \times 1) + (1 \times 2) + (1 \times 3) = 4 + 2 + 3 = 9$ cicli

Pertanto possiamo concludere che la sequenza 2 è più veloce, anche se richiede l'esecuzione di un'istruzione in più. Dato che tale sequenza richiede meno cicli di clock ma ha un'istruzione in più, deve avere un CPI più basso. Si può calcolare il valore del CPI come:

$$\text{CPI} = \frac{\text{Cicli di clock della CPU}}{\text{Numero di istruzioni}}$$

$$\text{CPI}_1 = \frac{\text{Cicli di clock della CPU}_1}{\text{Numero di istruzioni}_1} = \frac{10}{5} = 2{,}0$$

$$\text{CPI}_2 = \frac{\text{Cicli di clock della CPU}_2}{\text{Numero di istruzioni}_2} = \frac{9}{6} = 1{,}5$$

David A. Patterson, John L. Hennessy

Struttura e progetto dei calcolatori

Progettare con RISC-V

Edizione italiana a cura di Alberto Borghese

Fino a poco tempo fa i programmatori potevano fare affidamento sul lavoro dei progettisti di architetture e di compilatori e su quello dei produttori di chip per rendere più veloci e più efficienti a livello energetico i propri programmi senza il bisogno di apportare alcuna modifica. Questa epoca è finita: affinché un programma possa essere eseguito più velocemente deve diventare un **programma parallelo**.

La tecnologia moderna richiede che i professionisti di ogni settore dell'informatica conoscano sia il software sia l'hardware, la cui interazione ai vari livelli offre la chiave per capire i princìpi fondamentali dell'elaborazione. Per questo motivo gli autori di *Struttura e progetto dei calcolatori* hanno posto l'enfasi sulla relazione tra hardware e software, e il recente passaggio dalle architetture uniprocessore ai multiprocessori multicore ha confermato quanto la prospettiva del parallelismo sia giusta.

La novità di questa edizione è la scelta di trattare l'**architettura RISC-V**. Sviluppato inizialmente a Berkeley e progettato per funzionare con cloud computing, dispositivi mobili e altri sistemi embedded, questo insieme di istruzioni è più semplice ed elegante dell'insieme di istruzioni MIPS e presenta anche il vantaggio di non essere un'architettura proprietaria. Esistono quindi simulatori, compilatori e debugger RISC-V open source facilmente reperibili, e persino implementazioni RISC-V open source scritte nei linguaggi di descrizione dell'hardware. Gli obiettivi principali del corso sono:

- dimostrare con esempi concreti quanto sia importante comprendere il funzionamento dell'hardware, per ottenere buone prestazioni ed elevata efficienza energetica;
- evidenziare i temi principali di ogni argomento inserendo a margine del testo le icone associate alle «otto grandi idee» nella progettazione delle architetture;
- proporre nuovi esempi che riflettano il ricambio generazionale avvenuto nel passaggio dall'era dei PC all'era postPC (tablet, cloud, ARM, x86);
- distribuire il materiale relativo all'I/O in tutto il libro anziché racchiuderlo in un unico capitolo;
- aggiornare il contenuto tecnico per rispecchiare i cambiamenti avvenuti nell'industria, facendo riferimento, per esempio, ad architetture come il Cortex A-53 ARM e il Core i7 Intel.

David A. Patterson ha insegnato Computer Science alla University of California, Berkeley. È stato presidente della Association for Computer Machinery e consulente del presidente degli Stati Uniti per le tecnologie informatiche.

John L. Hennessy è rettore della Stanford University in California, dove insegna Computer Science dal 1977.

Il titolo originale dell'opera è
Computer and Organization Design RISC-V Edition, 1e.
Questa traduzione è pubblicata con l'autorizzazione di

ELSEVIER

Le risorse multimediali

online.universita.zanichelli.it/patterson-risc
A questo indirizzo sono disponibili le risorse multimediali di complemento al libro. Per accedere alle risorse protette è necessario registrarsi su **my.zanichelli.it** inserendo la chiave di attivazione personale contenuta nel libro.

Libro con ebook

Chi acquista il libro può scaricare gratuitamente l'**ebook**, seguendo le istruzioni presenti nel sito. L'ebook si legge con l'applicazione Booktab Z, che si scarica gratis da App Store (sistemi operativi Apple) o da Google Play (sistemi operativi Android).