

Architettura dei Sistemi di Elaborazione

Capitoli:

1. Introduzione
 - a. Approccio Strutturale
 - b. Pietre miliari
 - c. Tipologia di computer
 - d. Esempi di famiglie di computer
 - e. Unità metriche
2. Organizzazione sistemi di calcolo
 - a. Processori
 - b. Memoria principale
 - c. Memoria secondaria
 - d. Input/Output
3. Livello logico digitale
 - a. Porte logiche e algebra di Boole
 - b. Circuiti logici digitali elementari
 - c. Memoria
 - d. Chip della CPU e bus
 - e. Esempi di chip della CPU
 - f. Esempi di bus
 - g. Interfacce
4. Livello di microarchitettura

- a. Esempio di microarchitettura
 - b. Esempio di ISA: IJVM
5. Livello di macroarchitettura
- a. Overview del livello ISA
 - b. Tipi di dati
 - c. Formati di istruzioni
 - d. Indirizzamento
 - e. Tipi di istruzioni
 - f. Controllo del flusso
 - g. Architettura IA-32 IA-64
6. Linguaggio Assemblativo
- a. Introduzione al linguaggio assemblativo
 - b. Le macroistruzioni
 - c. Il processo di assemblaggio
 - d. Collegamento e caricamento
7. Architetture per il calcolo parallelo
- a. Classificazione di Fynn
 - b. Parallelismo dei chip
 - c. Coprocessori
 - d. Multiprocessori
 - e. Tipi di sistema multiprocessore
 - f. Multicomputer
 - g. Virtualizzazione

1. Introduzione

1.a Approccio Strutturale

Prefazione

Il computer può essere visto come una gerarchia di livelli, in ciascuno realizza una particolare e ben definita funzione.

Un calcolatore digitale è una macchina in grado di svolgere dei compiti eseguendo le istruzioni che le vengono assegnate. Il **programma** è una sequenza d'istruzioni. I circuiti elettronici dei computer possono riconoscere ed eseguire direttamente soltanto un insieme limitato d'istruzioni semplici in cui tutti i programmi devono essere convertiti prima di poter essere eseguiti. Tra queste troviamo:

- somma due numeri
- controllare se un numero vale zero
- copiare una porzione di dati da una parte all'altra della memoria.

L'insieme di queste istruzioni primitive forma il **linguaggio macchina** l'unico in grado di comunicare con il computer (stringhe di 0 e 1). Per poter progettare i computer in modo più organizzato i computer vengono strutturati come una serie di livelli di astrazione, cioè tramite un **approccio strutturale**. In questo modo la complessità del linguaggio macchina è gestita più agevolmente.

Linguaggi, livelli e macchine virtuali

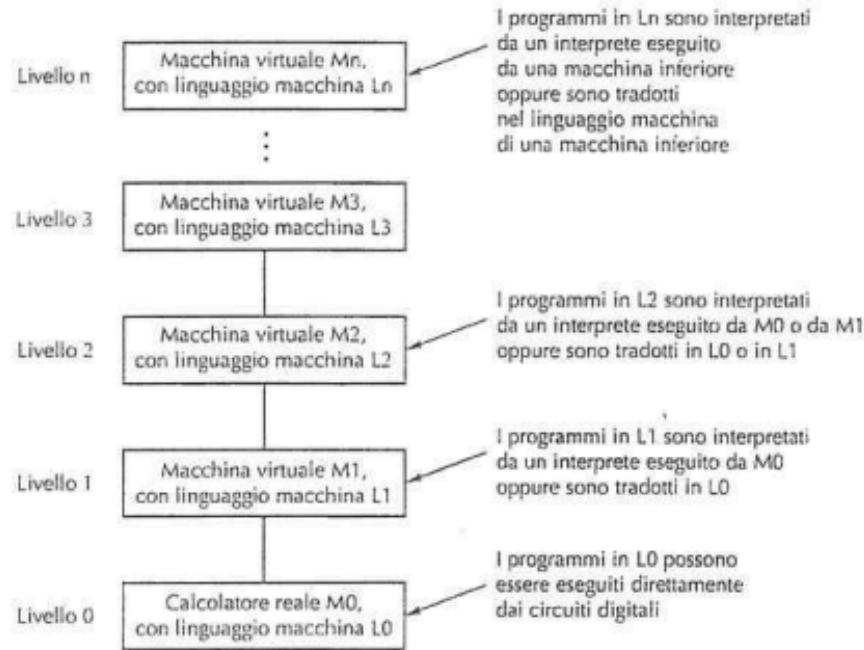
Un metodo per eseguire un programma scritto nel linguaggio L1 consiste nel sostituire, in una fase iniziale, ogni sua istruzione con un'equivalente sequenza di istruzioni in L0 (linguaggio macchina). Questa tecnica è chiamata **traduzione**.

L'altra tecnica consiste invece nello scrivere un programma in L0 che accetta come dati d'ingresso programmi in L1; tale programma li esegue esaminando un'istruzione alla volta e sostituendola direttamente con l'equivalente sequenza di istruzioni L0.

Questa tecnica è chiamata **interpretazione**.

Nel primo caso il programma viene convertito nel secondo invece viene esaminata, decodificata e eseguita ogni istruzione.

Le **macchine virtuali** non sono altro che ipotetici computer. Realizzare un computer che comprende un linguaggio più vicino all'utente è troppo costoso o complicato. Perciò si definiscono una serie di linguaggi, ciascuno dei quali più pratico rispetto al precedente, finché non se ne ottenga uno ottimale.



Attuali macchine multilivello

La maggior parte dei moderni computer consiste di due o più livelli. Il livello 0, che si trova alla base, rappresenta il vero e proprio hardware della macchina, i cui circuiti eseguono programmi scritti nel linguaggio macchina del livello 1. Sotto al livello 1 c'è il **livello dei dispositivi**, in questo livello i progettisti hanno a che fare con i singoli transistor.



Nel livello **logico digitale** vengono analizzate le **porte** (gate). Queste, pur essendo costruite utilizzando componenti analogici, possono essere modellate come dispositivi digitali. Ciascuna porta ha uno o più input digitali (0 o 1) e calcola in output una semplice funzione degli input, per esempio AND od OR. Combinando un piccolo numero di porte si può formare una memoria a 1 bit in gruppi. Ciascun **registro** può contenere un numero il cui valore può variare fino a un certo limite.

Nel livello di **micro-architettura** è presente una memoria locale, formata da un gruppo di registri (8-32) e un circuito chiamato **ALU** (*Arithmetic Logic Unit*), capace di effettuare semplici operazioni aritmetiche. I registri sono connessi alla ALU per formare un percorso dati. In alcune macchine le operazioni del percorso dati sono controllate da un programma chiamato **microprogramma**, mentre il percorso dati è controllato direttamente dall'hardware.

Il livello **ISA** (*Instruction Set Architecture*), vengono descritte l'insieme di istruzioni della macchina, istruzioni eseguite in modo interpretato dal microprogramma o dai circuiti elettronici, è presente un livello per interprete.

Il livello **macchina** è generalmente un livello ibrido, la maggior parte delle sue istruzioni fanno parte anche del livello ISA, ma ci sono nuove istruzioni, una diversa organizzazione della memoria e la capacità di eseguire programmi in modo concorrente. I nuovi servizi sono realizzati da un interprete eseguito dal livello 2,

chiamato sistema operativo.

Tra i livelli 3 e 4 vi è una divisione, i tre livelli inferiori non sono progettati per essere utilizzati dal programmatore medio, ma per eseguire interpreti e traduttori necessari come supporto per i livelli più alti. Questi ultimi sono scritti dai **programmatori di sistema**. I livelli superiori al 4 a differenza da quelli inferiori vengono generalmente tradotti e non interpretati. Inoltre i linguaggi inferiori sono numerici mentre invece a partire dal livello 4 i linguaggi contengono parole e abbreviazioni.

Il livello del linguaggio **assemblativo** fornisce ai programmati un linguaggio per scrivere programmi per i livelli 1, 2 e 3, il programma che esegue la traduzione è chiamato **assemblatore**.

Il livello del linguaggio **orientato al tipo di programma** consiste nei linguaggi ad **alto livello**, definiti per essere utilizzati dai programmati di applicazioni. Questi linguaggi vengono tradotti al livello 3 o 4 da un **compilatore**, o vengono interpretati.

Ciascun livello rappresenta una diversa astrazione, caratterizzati dalla presenza di oggetti e operazioni diverse. L'insieme dei tipi di dati, delle operazioni e delle funzionalità di ciascun livello è chiamato **architettura**.

Evoluzione delle macchine multilivello

I programmi scritti in linguaggio macchina possono essere eseguiti direttamente dai circuiti del computer, questi ultimi insieme alla memoria e ai dispositivi input/output formano l'**hardware** del computer. L'hardware consiste in oggetti tangibili. Al contrario il **software** consiste di **algoritmi** e programmi che possono essere memorizzati su diversi supporti HDD, CD, etc..

1.b Pietre miliari nell'architettura dei computer

I computer si sono evoluti nel tempo.

Generazione zero - Computer meccanici (1642-1945)

La prima persona a costruire una macchina calcolatrice funzionante fu lo scienziato francese Blaise Pascal (1623-1662), Pascal lo costruì a 19 anni, per aiutare il padre esattore fiscale. La macchina era costituita di ingranaggi e veniva azionata da una manovella capace di fare solo addizioni e sottrazioni, 30 anni dopo Leibniz riuscì ad eseguire anche moltiplicazioni e divisioni. Poi Charles Babbage (1792-1871) professore

di matematica della Cambridge University, oltre a inventare il tachimetro inventò anche la *macchina differenziale* pensata per eseguire un solo algoritmo, il metodo matematico delle differenze finite, nel quale i risultati venivano incisi su una lastra di rame. Costruì successivamente la *macchina analitica* composta da magazzino (memoria), mulino (unità computazionale), dispositivo di input (schede perforate), e di output (stampato e perforato). Negli anni '30 poi Konrad Zuse, costruì una serie di macchine calcolatrici automatiche utilizzando dei relè elettromagnetici. Non molto tempo dopo John Atanasoff e George Stibitz crearono una macchina basata sull'aritmetica binaria e utilizzava dei condensatori per la memoria. Nel '44 Aiken completò il Mark I, e successivamente il Mark II dando il via all'era dell'elettronica.

Prima generazione - Valvole (1945-1955)

Lo stimolo per lo sviluppo dei computer elettronici venne dalla Seconda Guerra Mondiale. I tedeschi usavano messaggi codificati per mezzo del dispositivo **Enigma**, in contrasto a questa dispositivo Alan Turing costruì il computer chiamato **Colossus**, il primo elaboratore digitale. Durante la guerra fu anche costruito il **ENIAC** (*Electronic Numerical Integrator And Computer*), questo elaboratore fu costruito da Mauchley e Eckert con 18000 valvole termoioniche e 1500 relè pesando oltre 30 tonnellate. Mentre entrambi erano impegnati a costruire **EDVAC** (*Electronic Discrete Variable Automatic Computer*) John von Neumann costruì la sua propria versione, la macchina **IAS**. Il primo progetto elementare che descrisse è conosciuto come **macchina di von Neumann**, era composta da cinque componenti principali: la memoria, l'unità aritmetico-logica, l'unità di controllo e i dispositivi di input e output. L'unità aritmetico-logica e l'unità di controllo formavano il cervello del computer, ad oggi è la **CPU** (*Central Processing Unit*). All'interno dell'unità aritmetico-logica vi era un registro a 40 bit chiamato **accumulatore**.

Seconda generazione - Transistor (1955-1965)

Il transistor fu inventato nel 1948 da Bardeen, Brattain, Shockley, e rivoluzionò i computer a tal punto che le valvole divennero obsolete. Il primo computer con i transistor fu chiamato **TX-0** (*Transistor eXperimental computer 0*). Ma la prima macchina che prese il volo fu PDP-1. Pochi anni dopo il PDP-8 introdusse un unico bus, chiamato *omnibus*, il **bus** è un insieme di cavi paralleli utilizzati per connettere i diversi componenti di un computer. Quest'architettura fu un grande passo in avanti. I progettista del computer 6600, Seymour Cray, dedicò la sua intera vita a costruire

supercomputer. Mentre le macchine descritte si concentravano esclusivamente sull'hardware i progettisti del B5000 costruirono una macchina con il preciso intento di programmarla, nacque così l'idea che anche il software ha la sua importanza.

Terza Generazione - Circuiti Integrati (1965-1980)

Nel 1958 l'invenzione dei circuiti integrati su silicio da parte di Kilby e Noyce permise di realizzare su un unico chip decine di transistor, ciò rese possibile costruire computer più piccoli, veloci ed economici. IBM realizzò così il System/360 progettata per i calcoli scientifici e commerciali. Una grande innovazione della serie 360 fu la **multiprogrammazione**, grazie a cui è possibile avere più programmi in memoria allo stesso tempo. Il 360 fu anche il primo a essere capace di emulare altri computer e risolse il dilemma dell'aritmetica binaria parallela e di quella decimale seriale. Nel mondo dei microcomputer invece fece un grande passo la serie PDP-11.

Quarta generazione - Integrazione a grandissima scala (1980-???)

Negli anni '80 la tecnologia **VLSI** (*Very Large Scale Integration*) permise di inserire in un unico chip milioni di transistor. Prima i computer erano talmente grandi e costosi che per farli funzionare esistevano speciali uffici chiamati **centri di calcolo**, ma dal 1980 i prezzi crollarono e anche i privati potevano permettersi di possedere un computer dando così vita al personal computer. I PC furono usati per l'elaborazione di testi, videogiochi, etc.. Uno dei primi fu l'Apple progettato da Steve Jobs e Steve Wozniak, ma anche IBM da lì a poco entrò nel mercato dei pc e con la CPU Intel 8088 divenne subito il computer più venduto della storia. IBM fece però lo sbaglio di rendere i progetti del computer dando il via a una catena di **cloni** del pc. IBM diventò talmente grande da schiacciare altri computer, uno dei sopravvissuti fu il Macintosh successore di Apple Lisa il primo computer dotato di **GUI** (*Grapical User Interface*). Alla metà degli anni '80 iniziò ad affermarsi un nuovo tipo di progettazione chiamata RISC, che consisteva nel sostituire le complicate architetture esistenti con altre molto più semplici e veloci. Queste macchine erano in grado di eseguire più istruzioni allo stesso tempo. Fu poi costruito il **FPGA** (*field-programmable gate array*) contenente una grande quantità di porte logiche. Nel 1992 fu creata la prima macchina a 64 bit di tipo RISC, per tutti gli anni '90 i sistemi diventarono sempre più veloci e ottimizzati.

Quinta generazione - Computer a basso consumo e computer invisibili

I computer iniziarono a rimpicciolirsi e nel 1989 fu rilasciato il primo tablet computer, il GridPad. In seguito altre macchine della stessa tipologia, ora chiamate **PDA** (*Personal Digital Assistans*) hanno avuto un'enorme diffusione. L'evoluzione di questi dispositivi ha portato allo smartphone. Il primo vero smartphone era chiamato Simon. Tuttavia oltre ai PDA vennero introdotti anche i computer ‘invisibili’, poichè integrati in elettrodomestici, orologi, etc.. Questi computer integrati hanno l’hardware e software **coprogettati**.

Se la prima generazione è rappresentata dalle macchine a valvole (per esempio ENIAC), la seconda dalle macchine a transistor (per esempio, l’IBM 7094), la terza dalle macchine a circuiti integrati (per esempio, l’IBM 360) e la quarta dai personal computer (per esempio, le CPU Intel), la quinta generazione è caratterizzata più da un cambiamento di modello che da una specifica nuova architettura.

1.c Tipologie di computer

Forze tecnologiche ed economiche

L’industria dei computer avanza più velocemente di tutte le altre, la principale forza è la capacità di integrare sempre più transistor all’interno di chip, il loro spessore sarà spesso di pochi atomi. Gordon Moore fondatore di Intel, sottolineò come ogni generazione di chip veniva introdotta ogni 3 anni con una quantità di memoria 4 volte maggiore alla precedente, che il numero di chip in un transistor aumentava a velocità costante. Questa osservazione è conosciuta come la **legge di Moore**.

Questa legge ha creato quello che gli economisti chiamano un **ciclo virtuoso**.

Un altro fattore che guida lo sviluppo tecnologico è il software, all’inizio pesavano pochi KB quelli moderni occupano vari MB quelli futuri richiederanno GB e così via

Tipologie di computer

Richard Hamming osservò che un cambiamento della quantità di un ordine di grandezza produce un cambiamento della qualità. Il guadagno fornito dalla legge di Moore consiste di costruire computer sempre più potenti allo stesso prezzo.

Computer usa e getta

I chip all'interno di cartoline d'auguri sono un esempio. Il più importante sviluppo nell'area è rappresentato dai chip **RFID** (*Radio Frequency Identification*) che si caricano tramite impulsi radio sufficientemente a lungo da riuscire a trasmettere all'antenna il proprio numero identificativo.

Microcontrollori

I computer integrati in apparecchiature che non sono elaboratori vengono chiamati **microcontrollori** e sono presenti in elettrodomestici, armi, giochi etc. Ciascuno di essi è dotato di un processore, di una memoria e di capacità I/O. Un'esempio è Arduino.

Dispositivi mobili e da gioco

Le console dei videogiochi come la Play sono un esempio, sono normali computer dotati di speciali capacità grafiche e sonore, ma poco espandibili e forniti di software limitato.

Personal computer

Sono macchine che comprendono modelli desktop o portatili, dotati di GB di memoria, di sistemi operativi sofisticati e molte possibilità di espansione e di software disponibili.

Server

Versioni potenziate dei personal computer o delle workstation come server di rete, per reti locali o Internet. È più veloce e più grande, ha una maggiore memoria di massa e una connessione alla rete più veloce. I **Cluster** non sono altro che un insieme di queste macchine connesse fra loro per lavorare in modo congiunto su uno stesso problema, quelli di grandi dimensioni si trovano in appositi locali o edifici chiamati data center.

Mainframe

Sono potenti server che gestiscono un numero massiccio di transazioni, solo i **supercomputer** sono più potenti dei mainframe, ma alla stessa potenza computazionale di un data center costano di più.

1.d Esempi di famiglie di computer

Ci sono molte famiglie di computer, tra le più importanti troviamo le architetture: x86, ARM E AVR. La prima è presente in quasi tutti i personal computer e suoi sistemi server. L'architettura ARM domina invece il mercato mobile. L'architettura AVR è diffusa nei microcontrollori economici.

Introduzione all'architettura x86

Nel 1968 Robert Noyce, inventore dell'integrazione dei circuiti sul silicio, Gordon Moore, famoso per l'omonima legge e Arthur Rock, un imprenditore fondarono la Intel Corporation per produrre chip di memoria. Grazie a Ted Hoff, fu costruita la prima CPU a 4 bit su un chip il processore 4004.

8008, prima CPU a 8 bit avente come limite 16KB di memoria.

8086, prima CPU a 16bit su un solo chip.

8088, usato anche come CPU per il primo PC IBM.

80286, introduzione della modalità protetta.

80386, prima CPU a 32 bit.

80486 avente la **memoria cache** da 8KB integrata, usata per conservare all'interno o vicino alla CPU, le parole di memoria utilizzate con più frequenza.

Pentium, due pipeline, istruzioni **MMX** (*MultiMedia eXtension*) per l'elaborazione di audio e video.

Pentium Pro, cache integrata a due livelli.

Pentium II, Pentium Pro con istruzioni MMX.

Pentium III, aggiunge le istruzioni **SSE** (*Streaming SIMD Extensions*) per la grafica 3D.

Pentium 4, avente l'*hypertreading* che permetteva ai programmi di dividere il lavoro in due flussi da eseguire in parallelo.

Core Duo, avente due core in un singolo circuito stampato.

Core, avente architettura quad-core a 64 bit.

Core i7, avente un processore grafico integrato.

Introduzione all'architettura ARM

Nei primi anni '80, la società Acorn Computer decise di lavorare a una macchina per competere con il PC IBM, ispirati dal progetto Berkeley RISC, decisamente di costruire una propria CPU e la chiamarono **ARM** (*Acorn RISC Machine poi ribattezzata Advanced RISC Machine*), la prima architettura fece la sua apparizione nel personal computer

Acron Archimedes. Dopo il successo Apple contattò Acron per creare la nuova società Advanced RISC Machine. Il nuovo processore fu chiamato ARM 610. ARM poi collaborò con DEC per realizzare StrongARM. Uno dei progetti più importanti fu l'ARM7, utilizzava cache separate per istruzioni e dati, e incorporava un insieme di istruzioni a 16 bit. A differenza di molte società di computer, ARM non produce nessuno delle CPU ma crea progetti librerie e sistemi di sviluppo per ARM. Questa architettura ha avuto grande successo nel mercato dei dispositivi a basso consumo, mobili e integrati.

Introduzione all'architettura AVR

Questa architettura è utilizzata nei sistemi integrati di fascia bassa. Creata nel 1996 da Alf e Vegard che progettarono una CPU RISC a 8 bit chiamata **AVR** (*Alf Vegard RISC*), viene realizzata in tre classi di microcontrollori, tinyAVR, megaAVR e AVR XMEGA. Questi montano generalmente tre tipi di memoria: flash, EEPROM e RAM.

1.e Unità metriche

Dato che le dimensioni delle memorie sono in potenza di due K assume il significato di $2^{10} = 1024$ invece che di $10^3 = 1000$

Esp.	Valore esplicito	Prefisso	Esp.	Valore esplicito	Prefisso
10^{-3}	0,001	milli	10^3	1.000	Kilo
10^{-6}	0,000001	micro	10^6	1.000.000	Mega
10^{-9}	0,000000001	nano	10^9	1.000.000.000	Giga
10^{-12}	0,00000000001	pico	10^{12}	1.000.000.000.000	Tera
10^{-15}	0,00000000000001	femto	10^{15}	1.000.000.000.000.000	Peta
10^{-18}	0,000000000000000001	atto	10^{18}	1.000.000.000.000.000.000	Exa
10^{-21}	0,0000000000000000000001	zepto	10^{21}	1.000.000.000.000.000.000.000	Zetta
10^{-24}	0,0000000000000000000000000001	yocto	10^{24}	1.000.000.000.000.000.000.000.000	Yotta

2. Organizzazione dei sistemi di calcolo

Un calcolatore digitale è un sistema in cui processori, memorie e dispositivi periferici sono connessi tra loro

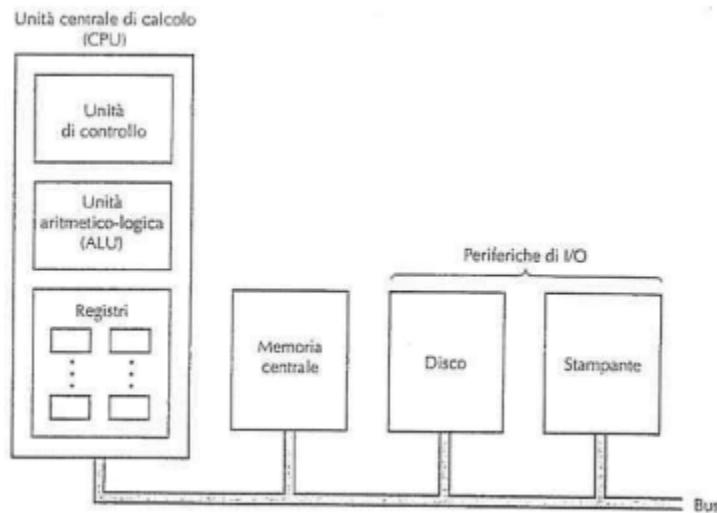
2.a Processori

La **CPU** (*Central Processing Unit*) è il cervello del computer e la sua funzione è quella di eseguire i programmi contenuti nella memoria principale prelevando le loro istruzioni, esaminandolo ed eseguendole una dopo l'altra.

La CPU contiene: l'unità di controllo (CU) e l'unità aritmetico-logica (ALU) la prima si occupa di prelevare le istruzioni dalla memoria principale e di determinare il tipo, mentre la seconda esegue le istruzioni.

La CPU contiene una piccola memoria ad alta velocità, utilizzata per memorizzare i risultati temporanei e alcune informazioni di controllo. Questa memoria è costituita da un certo numero di registri, ciascuno dei quali ha una funzione e dimensione predefinita. Il registro più importante è il **Program Counter (PC)**, che punta alla prossima istruzione da prelevare (fetch). Un altro registro importante è l'**Instruction Register (IR)**, che mantiene l'istruzione corrente in fase di esecuzione.

I componenti sono connessi fra loro mediante un bus, cioè un insieme di cavi paralleli sui quali vengono trasmessi indirizzi, dati e segnali di controllo.



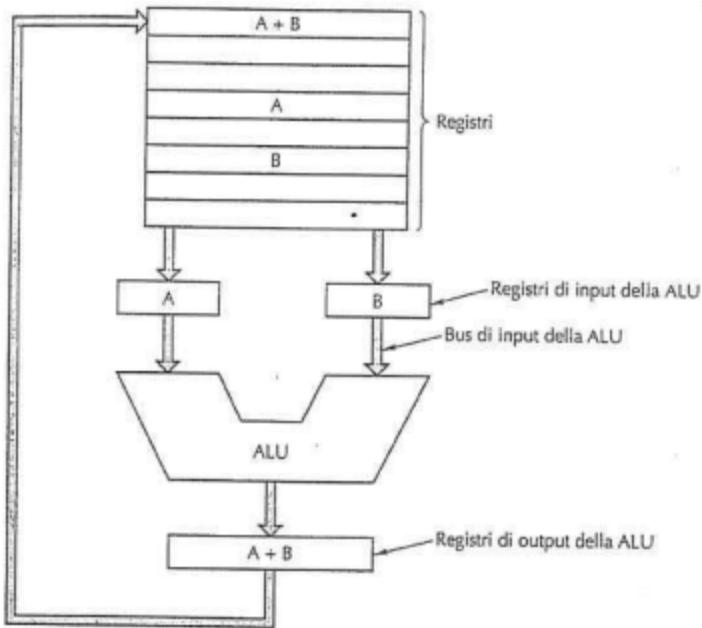
Organizzazione della CPU

Una tipica CPU di von Neumann contiene il **datapath** (o il percorso dati), composta da registri (1-32), dalla **ALU**, e vari bus che connettono tra loro le diverse parti. La ALU esegue, sui registri di input (A e B), addizioni sottrazioni e alcun operazioni semplici e il risultato è posto nel registro di output, che potrà poi essere immagazzinato in uno dei registri della CPU e successivamente nella memoria.

Le istruzioni possono essere:

registro-memoria permettono di prelevare parole di memoria per portarle all'interno dei

registri;
registro-registro dove gli operandi sono già presenti nei registri.



Esecuzione dell'istruzione

Il ciclo **fetch-decode-execute** è fondamentale per l'esecuzione delle istruzioni nel computer compiendo una serie di passaggi:

- 1) Preleva la successiva istruzione dalla memoria e la inserisce nell'IR;
- 2) Aggiorna il PC in modo che punti alla prossima istruzione;
- 3) Decodifica l'istruzione in IR;
- 4) se l'istruzione usa una parola in memoria, determinare dove si trova.
- 5) se necessario, prelevare la parola per portarla in un registro della CPU.
- 6) eseguire l'istruzione
- 7) tornare al punto 1 per iniziare l'esecuzione dell'istruzione successiva.

RISC contro CISC

In un'architettura RISC (*Reduced Instruction Set Computer*) la CPU è in grado di svolgere un insieme ridotto d'istruzioni, più veloci rispetto a quelle CISC.

In un'architettura CISC (*Complex Instruction Set Computer*) la CPU è in grado di comprendere istruzioni complesse nativamente, in grado di realizzare programmi più compatti che occupano minor spazio in memoria. A partire dal x486, le CPU Intel

contengono un sottoinsieme di istruzioni RISC che possono essere eseguite in un singolo ciclo nel datapath, mentre le altre complesse sono interpretate secondo la modalità CISC.

Principi di progettazione dei calcolatori

I progettisti di calcolatori devono tenere sempre d'occhio i cambiamenti tecnologici, perciò esiste una serie di **principi di progettazione RISC**:

Tutte le istruzioni sono eseguite direttamente dall'hardware: le istruzioni non devono essere interpretate. Per le architetture CISC, le istruzioni più complesse possono essere suddivise in parti ed eseguite come sequenze di microistruzioni.

Massimizzare la frequenza di emissione delle istruzioni: il parallelismo permette di eseguire più istruzioni contemporaneamente in modo da essere più veloce.

Le istruzioni devono essere facili da decodificare: le istruzioni dovrebbero essere regolari, di lunghezza predefinita, con un numero ridotto di campi/variabili.

Solo le istruzioni Load e Store fanno riferimento alla memoria: gli operandi vengono prelevati dai registri e memorizzati al loro interno e l'operazione di spostamento di questi ultimi può essere compiuta mediante apposite istruzioni.

Molti registri disponibili: dato che l'accesso in memoria è lento, devono essere forniti molti registri (almeno 32), così quando si fa la fetch di una word, questa può essere tenuta in un registro fino a che non è più utilizzata

Parallelismo a livello d'istruzione

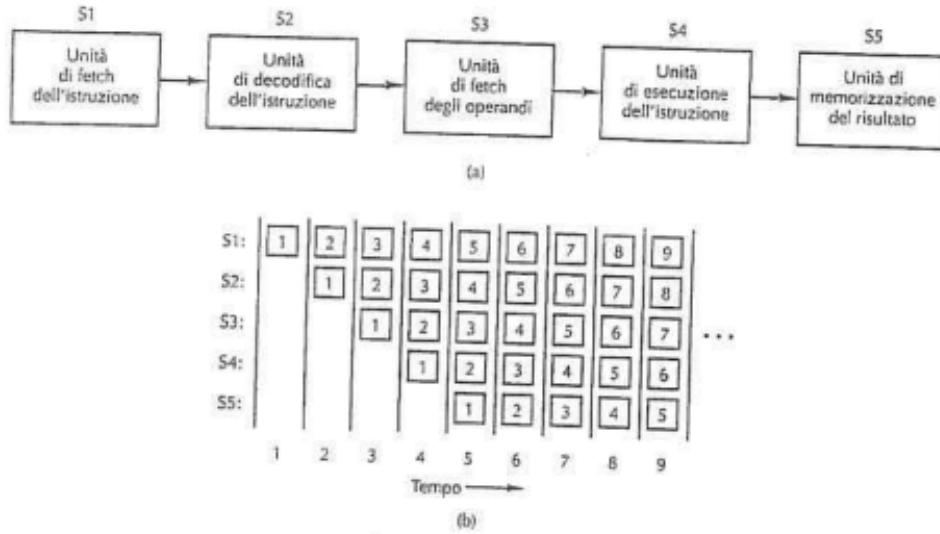
Poiché l'incremento del clock ha raggiunto un limite fisico si usa il parallelismo (compiere più istruzioni allo stesso tempo) per ottenere prestazioni più elevate. Si può ottenere in due forme diverse:

Parallelismo a livello di Istruzione: è sfruttato all'interno delle istruzioni per ottenere un maggior numero di istruzioni al secondo.

Parallelismo a livello di Processore: più CPU collaborano per risolvere lo stesso problema.

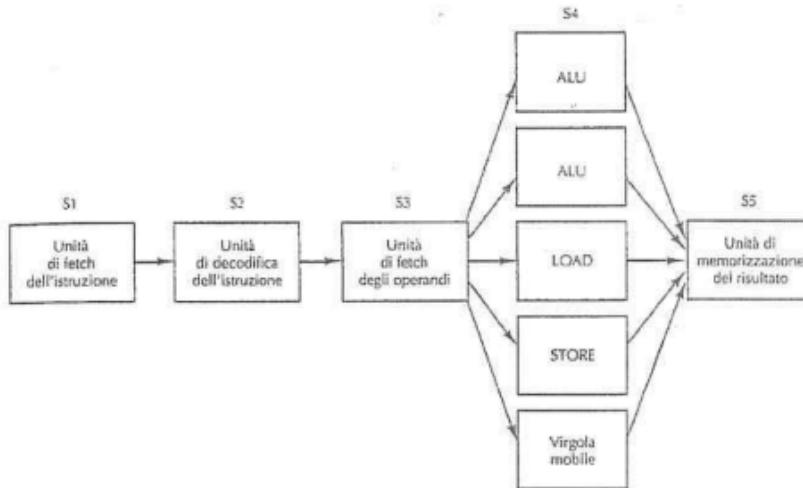
Pipelining: Uno dei grandi problemi è rappresentato dal fetch delle istruzioni dalla memoria, per ridurre questo problema inizialmente si è pensato di far in modo che la CPU avesse dei registri speciali (**prefetch buffer**) precaricati con l'istruzione da eseguire al momento dell'esecuzione. Con il concetto di **Pipeline** invece di dividere

l'esecuzione di un'istruzione solamente in due fasi, la si divide in un numero maggiore eseguite in parallelo ciascuna gestita da componenti hardware dedicati.



L'uso della pipeline permette di bilanciare la **latenza** (il tempo che un istruzione impiega per essere elaborata) e la **larghezza di banda del processore** (i MIPS della CPU).

Architetture superscalari: è ovvio che usare più pipeline consente performance maggiori rispetto ad utilizzarne una sola, ciò richiederebbe troppi componenti hardware però, perciò si usa una singola pipeline, alla quale vengono associate più unità funzionali.



Parallelismo a livello di processore

La richiesta di calcolatori sempre più veloci sembra inarrestabile, prima o poi ci si scontrerà con i problemi legati alla velocità della luce. I chip più veloci producono anche più calore, e la dissipazione costituisce un problema. Il parallelismo nel chip aiuta a migliorare le performance della CPU in parte, fino a 10 volte. L'unica soluzione è quella di progettare calcolatori con più CPU.

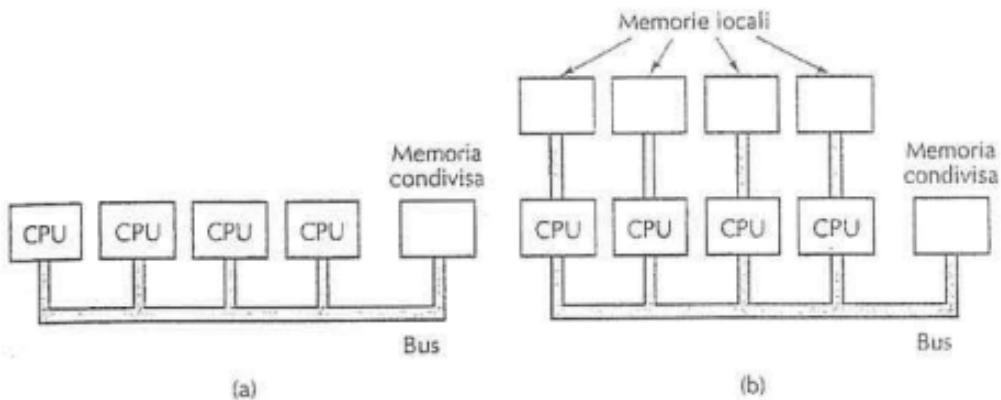
Computer con parallelismo sui dati: la regolarità e la struttura dei programmi che eseguono stessi calcoli ripetutamente li rende adatti a una esecuzione parallela. Esistono due modi per eseguire questi programmi in modo rapido ed efficiente: I processori SIMD: (*Single Instruction-stream Multiple Datastream*) consiste di un elevato numero di processori identici che eseguono la stessa sequenza d'istruzioni su insiemi diversi di dati.

I processori vettoriali: eseguono in modo efficiente una stessa sequenza di operazioni su coppie di dati, ma a differenza dei SIMD, tutte le operazioni di addizione sono eseguite da un unico sommatore strutturato a pipeline.

Entrambi lavorano su array di dati.

Multiprocessori: un sistema composto da più CPU con una memoria in comune, coordinati per evitare che si ostacolino a vicenda. Quando due o più CPU hanno la possibilità di interagire in modo così profondo si dice che sono *tightly coupled*.

Multicomputer: La difficoltà dei multiprocessori consiste nel connettere tutti i processori alla memoria (più di 256 CPU). Per aggirare questi problemi i progettisti hanno costruito sistemi composti da un gran numero di calcolatori interconnessi, ciascuno dotato di una memoria privata. Le CPU sono dette *loosely coupled* e comunicano attraverso lo scambio di messaggi, in architetture grandi la completa interconnessione non è fattibile così sono utilizzate topologie differenti.



2.b Memoria principale

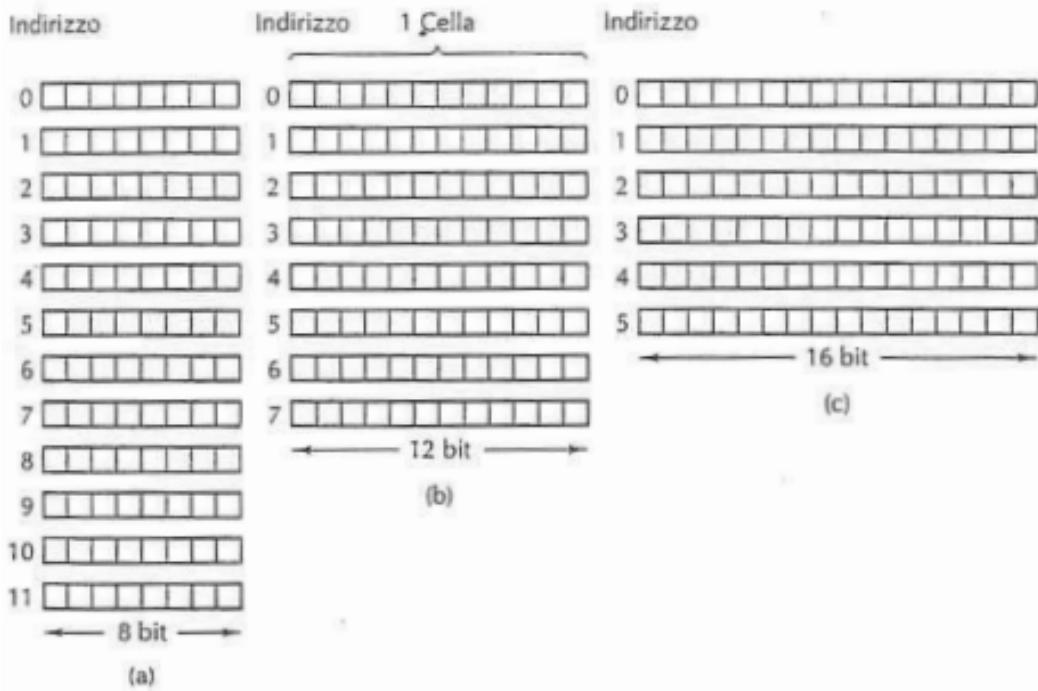
La memoria è quella parte del calcolatore in cui sono depositati programmi e dati (store o storage).

Bit

L'unità base della memoria è il **bit** (*BInary digiT*) può avere valore 0 o 1. Quando si dice che i calcolatori utilizzano l'aritmetica binaria perché è efficiente s'intende che l'informazione digitale può essere memorizzata utilizzando dei valori di una certa quantità fisica continua, come la tensione o la corrente. Nel formato decimale 16 bit possono memorizzare i numeri da 0 a 9999, permettendo solo 10.000 combinazioni, mentre una stringa di 16 bit può avere 65536 combinazioni. Un computer ragiona unicamente interpretando gruppi di bit, cioè comandi rappresentati da sequenze di 0 e 1 (00101100).

Indirizzi di memoria

Le memorie sono costituite da un certo numero di **celle**, ciascuna delle quali può memorizzare informazioni, identificabili da un numero chiamato **indirizzo** attraverso il quale il programma può riferirsi ad essa. Se una memoria ha n celle i suoi indirizzi varieranno da 0 a $n-1$.



come possono essere organizzati 96 bit di memoria

I calcolatori che usano il sistema numerico binario esprimono gli indirizzi di memoria in notazione binaria. Se un indirizzo ha m bit, il massimo numero di celle indirizzabili è 2^m . La cella rappresenta l'unità più piccola indirizzabile, 8 bit formano 1 **byte** e sono raggruppati in **parole**. Un calcolatore con parole a 32 bit ha 4 byte per parola, 64 bit ha 8 byte per parola. Una macchina a 32 bit avrà registri a 32 bit e istruzioni per manipolare parole a 32 bit, mentre una di 64 bit avrà registri a 64 bit e istruzioni per manipolare parole a 64 bit.

Ordinamento dei byte

I byte in una parola possono essere scritti da sinistra a destra (**Big endian**) e viceversa (**Little endian**).

Codici correttori

I computer possono, occasionalmente, commettere degli errori: al fine di rilevare e correggere l'eventuale errore si utilizzano dei codici di rilevazione e/o correzione degli errori aggiungendo dei bit extra a ogni parola.

Una **parola di codice** (o **codeword**) con n -bit è una parola che contiene m bit per i dati

e r per il controllo dell'errore ($n = m + r$).

La **distanza di Hamming** tra due parole di codice è data dal numero di differenze tra bit corrispondenti e ciò indica il numero di errori. Per il calcolo è sufficiente sommare il risultato dell'EXOR bit-a-bit delle due parole.

Come semplice esempio di codice a correzione di errore consideriamo un codice quale al dato si aggiunge un singolo **bit di parità**, scelto in modo che il numero di bit nella parola di codice sia pari oppure dispari. Un codice di questo tipo ha distanza 2, dato che ogni errore genera una parola di codice la cui parità è errata. Servono due errori singoli per passare da una parola di codice valida a un'altra anch'essa valida.

Immaginiamo di voler progettare un codice con m bit di dati e r di controllo, capace di correggere tutti gli errori singoli. Ciascuna delle parole di memoria legali ha n parole di codice illegali a distanza 1 da essa. Queste sono formate invertendo sistematicamente ciascuno degli n bit nella parola di codice generata. La percentuale di overhead decrementa al crescere della dimensione della parola.

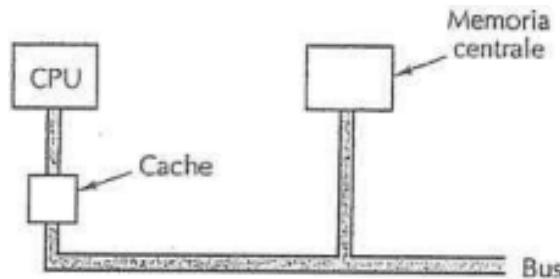
Dimensione parola	Bit di controllo	Dimensione totale	Percentuale di overhead
8	4	12	50
16	5	21	31
32	6	38	19
64	7	71	11
128	8	136	6
256	9	265	4
512	10	522	2

Memoria cache

Storicamente le CPU sono sempre state più veloci delle memorie, i miglioramenti di quest'ultime hanno contribuito a incrementare anche le prestazioni delle CPU. I progettisti delle memorie cercano di aumentare la capacità dei loro chip, non la loro velocità, aumentando lo squilibrio tra memoria e CPU. Quando la CPU lancia una richiesta alla memoria, essa otterrà la parola desiderata dopo svariati cicli di CPU. Per risolvere questo problema ci sono due metodi:

- far iniziare le istruzioni di lettura dalla memoria non appena vengono inco2ntrate, permettendo al contempo di continuare l'esecuzione e bloccando la CPU quando un'istruzione tenta di usare una parola di memoria non ancora arrivata, però più lenta e la memoria e più si verifica questo problema.
- richiedere ai compilatori di non generare codice che utilizzi parole ancor prima che

queste siano arrivate, ciò causa però un peggioramento nelle prestazioni. Quindi bisogna costruire delle memorie veloci (grandi) collocate sul chip della CPU (dato che utilizzare il bus di memoria è troppo lento) costerebbero troppo. Un riferimento di memoria può essere acceduto più volte (**principio di località temporale e spaziale**) quindi una soluzione può essere: L'utilizzo di una memoria piccola e veloce tra CPU e memoria chiamata **memoria cache** che memorizza piccole porzioni di dati. Le parole di memoria più utilizzate sono mantenute nella cache, così quando la CPU ha bisogno di una parola, la CPU guarda direttamente nella cache e poi nella memoria centrale.



Assemblaggio e tipi di memoria

Le memorie erano vendute e installate in un unico chip. Ora vari chip sono montati su una piccola scheda a circuiti stampati di 8 o 16 elementi, montati su piccoli circuiti stampati e vendute come singole unità. Questa unità è chiamata **SIMM** (*Single Inline Memory Module*) oppure **DIMM** (*Dual Inline Memory Module*) a seconda del numero di connettori allineati su uno o su due lati della scheda. Le SIMM trasferiscono 32 bit per ciclo di clock mentre le DIMM 64 bit.

2.c Memoria secondaria

La memoria centrale non è mai troppo grande.

Gerarchie di memoria

La dimensione della memoria è inversamente proporzionale al tempo di accesso e direttamente al costo. I registri della CPU sono le memorie più piccole, più veloci e le

più costose, i nastri magnetici e i dischi ottici sono le memorie più lente ma le più economiche e grandi.



Dischi magnetici

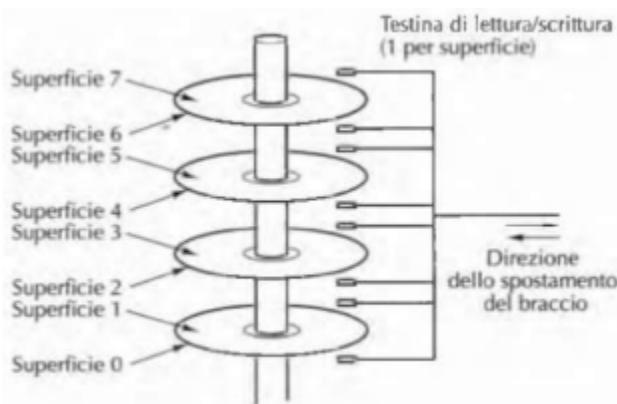
Un disco magnetico consiste di uno o più piatti di alluminio rivestiti con una superficie magnetizzabile ruota rispetto al proprio centro. Una testina che si muove modo longitudinale al disco, grazie al movimento di rotazione del disco, è in grado di coprirne l'intera superficie. La testina contiene un **solenoid**e che è in grado di orientare le particelle di materiale ferroso a seconda della polarità della corrente che attraversa la spirale. La sequenza circolare dei bit scritti in una circonferenza completa è detta **traccia** del disco. Le tracce sono divise in un certo numero di **settori** di lunghezza fissa e preceduta da un **preambolo** che permette alla testina di sincronizzarsi e un codice per la correzione di errore, un codice Hamming oppure il codice **Reed-Solomon**. Tra due settori consecutivi vi è una piccola area chiamata **intersector gap**.



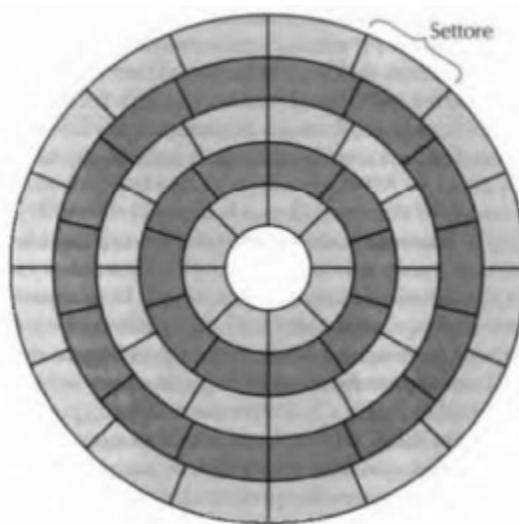
Questi dischi vengono chiamati anche **dischi Winchester** e il primo aveva 30MB di memoria. La maggior parte dei dischi consiste di più piatti impilati verticalmente in cui ciascuna superficie ha il proprio braccio e la propria testina. La sua performance dipende da:

tempo medio di seek: il posizionamento della testina nel raggio della traccia ricercata.
latenza rotazionale: il tempo necessario al disco per posizionare il corretto settore sotto la testina.

dal tempo di trasferimento: dipende dalla densità lineare e dalla velocità di rotazione.



L'insieme di tracce alla stessa distanza dal centro è chiamato **cilindro**. Nei nuovi dischi il numero di settori per traccia aumenta in corrispondenza delle zone più esterne, ogni disco ha un processore dedicato chiamata **disk controller** che, oltre a pilotare il dispositivo, accetta i comandi software, corregge i dati, etc..



Un esempio di dischi magnetici sono: floppy disk, hard disk

Dischi IDE

Nei primi PC il sistema operativo leggeva e scriveva dati sul disco inserendo parametri nei registri della CPU e poi invocando il **BIOS** (*Basic Input Output System*) che era memorizzato nella ROM interna, esso lanciava le istruzioni macchina necessarie per caricare i registri del controllore, che iniziava il trasferimento dati. La tecnologia evolse rapidamente e fu utilizzato lo standard **IDE** (*Integrated Drive Electronics*) in cui il controllo era strettamente integrato con l'unità, mentre prima si trovava su una scheda separata.

Lo standard IDE è poi evoluto nell'**EIDE** (*Extended IDE drives*) che supportavano anche un secondo schema d'indirizzamento chiamato **LBA** (*Logical Block Addressing*), che numerava i settori, aumentava la capacità massima e la velocità. I controllori EIDE potevano avere due canali su ciascuno dei quali poteva essere collegato un disco primario e secondario, questa organizzazione ammetteva un massimo di quattro unità da controllare.

Il successore dell'EIDE fu chiamato **ATA-3** (*Advanced Technology for Attachment-3*) aumentò la velocità, il successivo standard fu chiamato **ATAPI-4** (*ATA Packet Interface-4*) e i seguenti incrementarono la dimensione e la relativa velocità di trasferimento: **ATAPI-5, ATAPI-6**.

Lo standard successivo **ATAPI-7**, invece di incrementare la dimensione del connettore per aumentare la banda, utilizza un serial **ATA** per trasferire i bit su un connettore a 7 pin, sostituendo il connettore piatto a 80 pin paralleli.

Dischi SCSI

Nel 1986 **SCSI** (*Small Computer System Interface*) divenne standard ANSI. I dischi SCSI hanno una organizzazione del tutto simile ai dischi IDE ma necessitano di differenti interfacce ed hanno più alte velocità di trasferimento.

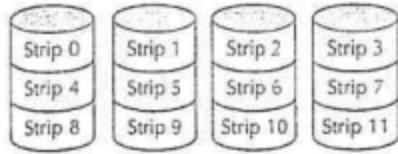
Nome	Bit di dati	MHz del bus	MB/s
SCSI-1	8	5	5
Fast SCSI	8	10	10
Wide Fast SCSI	16	10	20
Ultra SCSI	8	20	20
Wide Ultra SCSI	16	20	40
Ultra2 SCSI	8	40	40
Wide Ultra2 SCSI	16	40	80
Ultra3 SCSI	8	80	80
Wide Ultra3 SCSI	16	80	160
Ultra4 SCSI	8	160	160
Wide Ultra4 SCSI	16	160	320
Wide Ultra5 SCSI	16	320	640

Questi dischi sono lo standard in workstation e server di fascia alta. SCSI non è soltanto un'interfaccia per hard disk, ma anche un bus dove possono essere collegati fino a 7 dispositivi. Ciascun dispositivo SCSI possiede un ID, due connettori, uno per l'input e l'altro per l'output. I controllori SCSI possono funzionare come iniziatore o come destinatari di comunicazione.

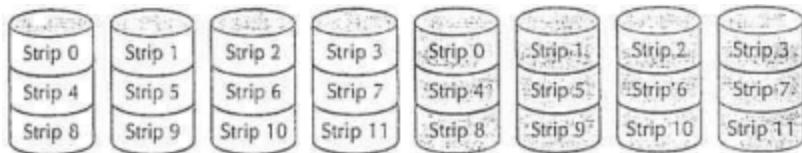
RAID

Le prestazioni delle CPU sono incrementate esponenzialmente nell'ultimo decennio, non è accaduto lo stesso per i dischi. Per migliorare le performance dei dischi si pensò di adottare la medesima strategia del calcolo parallelo, questa idea fu adottata e portò a una nuova classe di dispositivi di I/O chiamata **RAID** (*Redundant Array of Inexpensive Disks*), in opposizione agli **SLED** (*Single Large Expensive Disk*). L'idea è di far vedere al calcolatore il sistema RAID, costituito da un insieme di dischi, come un unico enorme disco virtuale con elevata performance ed affidabilità, cioè come uno SLED. Tutti i RAID hanno la proprietà di distribuire i dati sulle diverse unità per permettere la gestione parallela. Sono possibili differenti schemi: da livello 0 fino a livello 5.

RAID 0: i dati sono suddivisi in strisce di k settori e memorizzati in dischi diversi con modalità ciclica (round-robin), il blocco di dati può essere letto con 4 letture parallele. Questo schema lavora meglio quando le richieste sono di grandi dimensioni, non è un vero raid perché non esiste ridondanza (quando un disco si rompe tutti i dati sono persi completamente) Inoltre ha minore affidabilità di un sistema SLED.



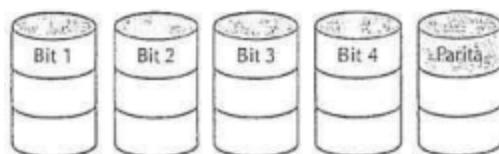
RAID 1: reitera il medesime funzionamento del RAID 0 duplicando i dischi, così ci sono tanti dischi primari quanti dischi di backup, durante una scrittura ogni strisci è scritta due volte (in parallelo), durante una lettura possono essere utilizzate tutte le copie, distribuendo il carico su più dischi. Le prestazioni sono uguali per la scrittura, migliorate per la lettura rispetto a un sistema SLED, se un disco si rompe si può utilizzare una sua replica.



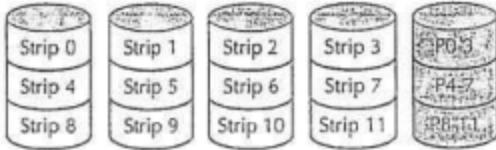
RAID 2: utilizza le parole binarie, oppure i byte, per decomporre le informazioni sui vari dischi. Con 7 dischi è possibile dividere un bit per disco, i byte si dividono in nibble (4 bit) e si aggiungono 3 bit ottenendo un codice di Hamming. Se un disco si rompe, non ci sono problemi poiché si può costruire una copia facilmente. Le operazioni sono in parallelo.



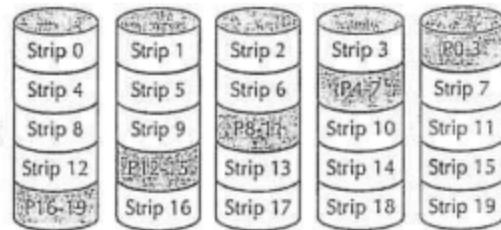
RAID 3: Versione semplificata del RAID 2, un singolo bit di parità è scritto in un disco di parità separato per ogni parola. Tutti i dischi devono essere sincronizzati. Non offre garanzie di affidabilità su errori casuali perché un solo bit non è sufficiente, se si rompe un disco è possibile ricostruire il suo contenuto.



RAID 4: lavora con le strisce e non richiede dischi sincronizzati, è come RAID 0 con una parità striscia per striscia scritta su disco separato di parità, si esegue l'EXOR bit-a-bit di tutte le strisce ottenendo così una striscia di parità.



RAID 5: lavora con le strisce e non richiede dischi sincronizzati, distribuisce in modalità round-robin le strisce di parità. Se si rompe un disco il processo per ripristinarlo è complesso.



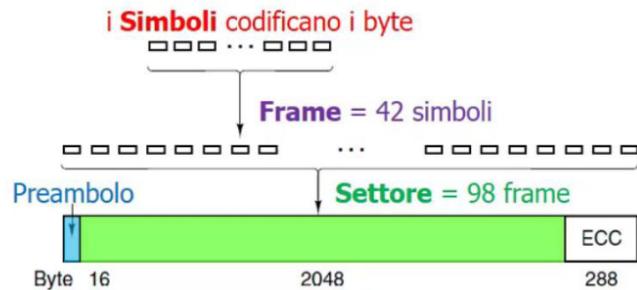
Dischi a stato solido (SSD)

Sono i dischi basati su memoria flash non volatile, la capacità di memorizzazione si degrada nel tempo a causa dell'usura dei transistor. Hanno prestazioni eccellenti, superiori ai dischi magnetici, ma costano molto e hanno una percentuale di fallimento più alta anche se sono più tolleranti alle vibrazioni in quanto non hanno elementi meccanici.

CD-ROM

Nel 1984, Philips e Sony svilupparono un supporto ottico in grado di memorizzare dati: il **CD-ROM** (*Compact Disc - Read Only Memory*). È fatto di policarbonato sul quale è depositato un sottile strato di alluminio riflettente e, poi, ricoperto da una vernice resistente. Le scanalature nel substrato di policarbonato sono chiamate **pit**, mentre le aree non incise **land**. La transizione da un pit a land può rappresentare un valore logico alto o basso, i pit sono scritti in modo continuo su una spirale che parte dal buco centrale del disco, per leggere un CD-ROM si usa un laser a bassa potenza. Il formato

base di un CD-ROM codifica ogni byte con un simbolo a 14-bit, un gruppo di 42 simboli consecutivi forma un frame e un settore è fatto di 98 frame.



Per memorizzare informazioni esistono due modalità: con correzione di errore oppure senza ECC. Un settore contiene: un preambolo, la sezione dei dati e il codice ECC. Il secondo modo è utilizzato per applicazioni audio/video.

CD-Registrabili

Fisicamente simili ai CD-ROM, hanno uno strato di pigmento che permette di scrivere i pit. Contengono una scanalatura che permette di guidare il laser in fase di scrittura. Quando il raggio laser colpisce il pigmento, rompe un legame molecolare creando una regione oscura non ripristinabile.

Anche noti come **CD-RW** (*CD - ReWritable*), al posto del pigmento troviamo una lega che ha due stati stabili: cristallino e amorfo. Il laser utilizza 3 diverse intensità.

DVD

I DVD sono progettati in modo simile ai CD ma sono più innovativi. Hanno pit più piccoli, una più stretta spirale, laser rosso, maggiore capacità e maggior throughput e sono stati definiti quattro formati: single-sided single-layer, single-sided dual-layer, double-sided single-layer, double-sided dual-layer.

Blu-ray

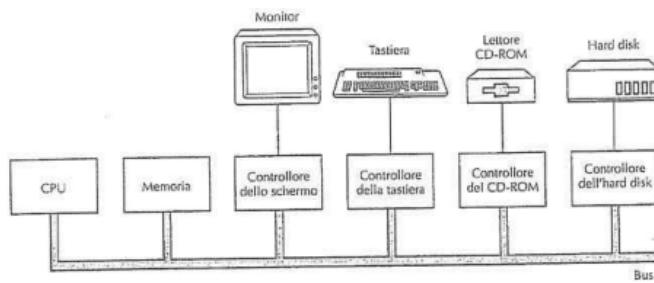
Uguali ai DVD ma utilizzano un laser rosso che permette una messa a fuoco più accurata e l'uso di pit e land più piccoli con conseguente più spazio.

2.d Input/Output

Come detto un calcolatore è composto da tre componenti principali: CPU, memorie, dispositivi di I/O.

Bus

La **scheda madre** (*motherboard*) è la scheda principale del computer che contiene la CPU, gli slot per la RAM, i bus di comunicazione e altri connettori per i controller dei dispositivi di I/O. All'interno di un computer il bus è un sistema di comunicazione utilizzato per collegare la CPU, la memoria e i dispositivi I/O, ciascuno dei quali è composto da due parti, una contenente la maggior parte dell'elettronica, chiamata **controllore**, e l'altra contenente il dispositivo stesso. Il connettore è connesso al suo dispositivo mediante un cavo che si collega al connettore nella parte posteriore della scatola meccanica.



Negli attuali computer i bus possono essere seriali o paralleli. Il bus interno è tipicamente parallelo: un insieme di cavi paralleli consente di trasmettere più bit nella stessa unità di tempo. I bus seriali hanno pochi cavi e trasmettono dati in istanti di tempo successivi. Il bus interno è suddiviso in:

Bus dati: trasmette informazioni tra componenti interni del computer.

Bus di controllo: si compone di segnali di controllo e sincronizzazione utilizzati per stabilire chi può trasmettere dati sul bus dati, per indicare il tipo di operazione, la dimensione dei dati trasmessi, la richiesta di interruzione, etc..

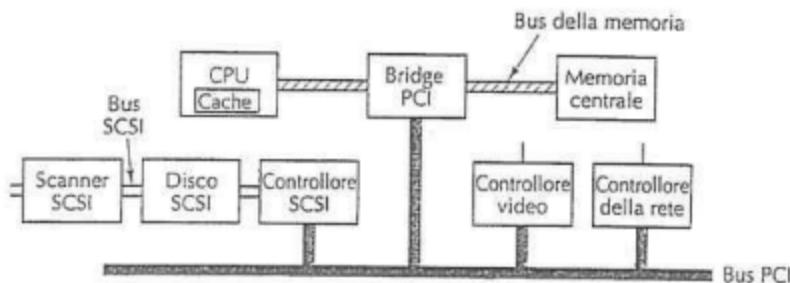
Bus indirizzi: specifica la posizione fisica dove i dati devono essere letti o scritti.

Il controllore gestisce il proprio dispositivo di I/O e interfaccia il bus in modo da trasferire i dati, i principali passi che seguono sono:

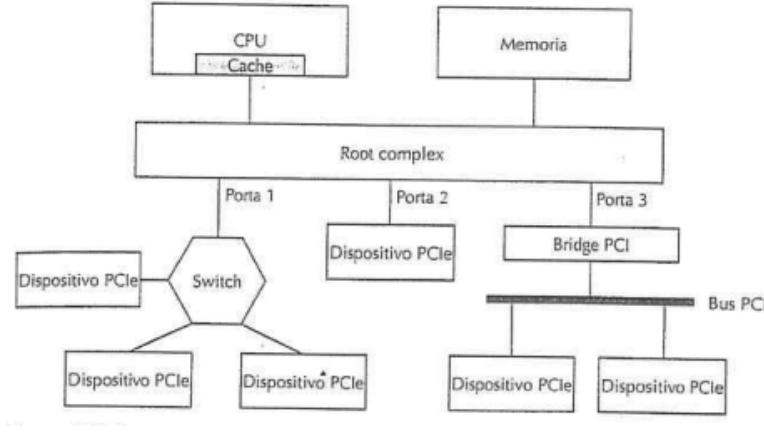
- il programma fornisce il comando al controller del disco,
- il controller comunica al drive di posizionarsi, con una seek, sulla traccia che contiene i dati da leggere,
- il drive invia la sequenza di byte al controller,

- il **DMA** (*Direct Memory Access*) è una tecnica che permette al controllore del disco di scrivere dati direttamente in memoria senza che la CPU intervenga nel processo. Quando il trasferimento è completato, il controller invoca un **interrupt** che forza la CPU a sospendere il programma corrente e avviare una procedura speciale (**interrupt handler**). Questa procedura è chiamata **ISR** (*Interrupt Service Routine*) ed ha il compito di verificare se ci sono stati errori e di comunicare al sistema operativo che il trasferimento è terminato. Quando la ISR è terminata, la CPU riprende il programma che aveva sospeso. Poiché il bus è una risorsa condivisa, sia la CPU sia i controllori di I/O potrebbero richiederne l'utilizzo, perciò un chip chiamato arbitro del bus, stabilisce, in base a delle priorità, chi è deputato ad utilizzarlo. Quando un controller di dispositivo richiede il bus, questo non gli viene sempre concesso (si perdono dati) rubando così dei cicli alla CPU (**cycle stealing**) e rallentandone le prestazioni. L'utilizzo di un solo bus condiviso ha funzionato funzionato fintato che i PC erano ben bilanciati. Il bus singolo del primo PC era il bus **ISA** (*Industry Started Architecture*), il successore fu l'**EISA**, oggi il più popolare è il bus **PCI** (Peripheral Component Interconnect) e per incoraggiare l'adozione di questo nuovo standard da parte del mercato, Intel decise di rendere di pubblico dominio tutti i brevetti.

L'evoluzione dei PC ha portato ad una soluzione che potesse gestire sia periferiche EISA che PCI, la CPU e la memoria comunicano con un bus dedicato, i dispositivi veloci connessi al bus PCI, mentre quelli vecchi all'EISA e il bridge collega i due bus.



L'evoluzione del bus PCI è il bus PCIe che utilizza una rete punto-punto con linee seriali di bit a commutazione di pacchetto.



Altri dispositivi di I/O

Poiché i computer sono macchine che possono eseguire compiti ripetitivi senza commettere errori, l'interazione con il mondo esterno è essenziali. I dispositivi esterni dovrebbero rappresentare informazioni in una forma comprensibile agli essere umani. Le periferiche esterne possono essere divise in tre categorie:

Input: inserire dati all'interno del computer,

Output: permettono di fornire dati al computer,

Input-output: fare entrambe.

Tastiere, Monitor, Mouse, Stampanti, Controller sono tutti dispositivi di I/O.

Apparecchiature per le telecomunicazioni

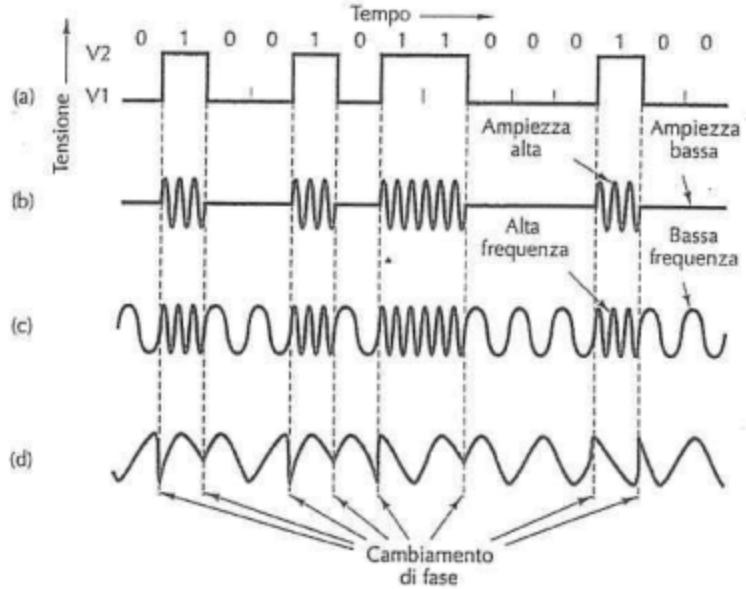
Tutte le persone che hanno un PC a casa ed utilizzano internet hanno un apparato per la connessione alla rete. Il segnale sinusoidale (detto **portante**) subisce una distorsione accettabile rispetto al doppino telefonico, e variando:

(b) Modulazione di ampiezza: si utilizzano 2 diverse tensioni (0,1)

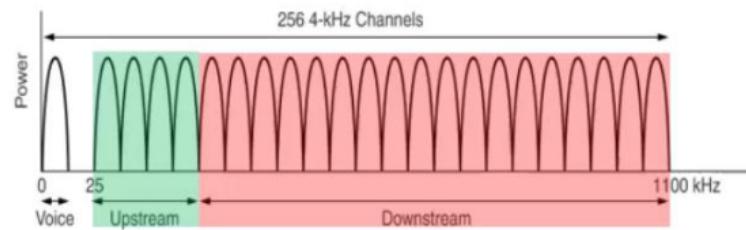
(c) Modulazione di frequenza: l'ampiezza è costante mentre varia la frequenza della portante (0,1)

(d) Modulazione di fase: l'ampiezza e la frequenza rimangono costanti mentre varia la fase (0,1)

è possibile trasmettere informazioni binarie. Un modem (modulator-demodulator) è un dispositivo che esegue la modulazione di un segnale digitale per la trasmissione e poi demodula il segnale analogico per la ricezione dei dati.



La DSL è una tecnologia che permette di superare le limitazioni dei modem, il più popolare è l'**ADSL** (Asymmetric DSL), la frequenza della banda del local loop dipende dalla lunghezza. Il canale 0 è utilizzato per il servizio telefonico tradizionale, i canali 1-5 non sono utilizzati, due canali sono utilizzati per il controllo del traffico in uscita e di quello in ingresso, 248 canali sono utilizzati per i dati.



Codifica dei caratteri

Poiché ogni computer poter ricever/fornire informazioni agli esseri umani è necessario che riconosca un insieme di caratteri costituito almeno dai simboli dell'alfabeto. I caratteri sono codificati con i numeri naturali, la corrispondenza tra caratteri e numeri è detta codice o **codifica dei caratteri**. I computer per poter comunicare tra loro devono utilizzare lo stesso codice di caratteri, per questa ragione sono stati definiti alcuni standard. Gli standard per i codici di caratteri sono:

ASCII: è il codice caratteri più diffuso ed utilizzato, ogni carattere ASCII è codificato con 7 bits, i codici iniziali vanno da 00 a 1F sono caratteri di controllo e non sono stampabili.

Funziona bene per il linguaggio anglosassone ma meno per altre lingue, dato che alcuni caratteri non sono presenti nella mappa ASCII, perciò è stata estesa aggiungendo un bit di codifica.

UNICODE: Un gruppo di aziende di computer decise di risolvere questi problemi formando un Consorzio per creare un nuovo sistemi di codifica chiamato UNICODE. L'idea è assegnare ad ogni carattere un valore a 16-bit unico, chiamato code point. Nonostante risolve molti problemi associati all'internalizzazione, esso non risolve problemi legati alla cultura locale: ordinamento degli ideogrammi nella mappa, nascita di nuovi ideogrammi, usa la stessa codifica per caratteri che sembrano uguali ma che non lo sono.

UTF-8: UNICODE ha presto esaurito i code point e inoltre spreca 16 bit per codificare la mappa ASCII per queste ragioni è stato sviluppato lo standard UTF-8 (*Universal Character Set Transformation Format*) hanno lunghezza variabile da 1 a 4 byte e possono codificare 2 miliardi di caratteri, è lo standard per WWW, nei bit più significativi del primo byte è iscritto il numero di byte necessari per la codifica: 0xxxxxx codice 1 byte, 110xxxx 2 byte, 1110xxx 3 byte, 11110xx 4 byte.

3. Livello logico digitale

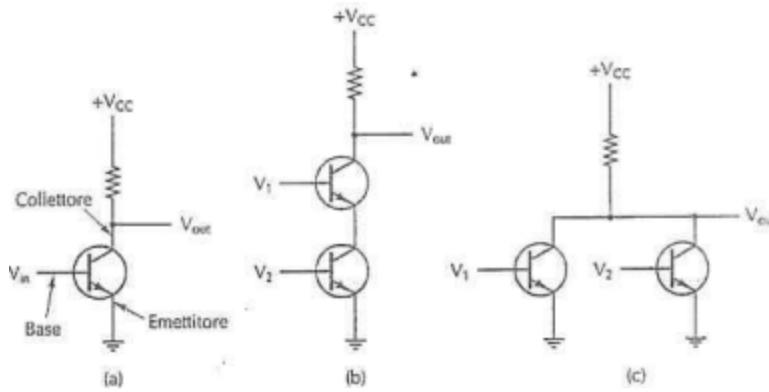
Gli elementi base a partire dai quali sono costruiti tutti i calcolatori digitali vengono rappresentati nel livello logico digitale, è fatto di piccoli dispositivi elettronici chiamate porte logiche che lavorano con l'algebra di Boole.

3.a Porte logiche e algebra di Boole

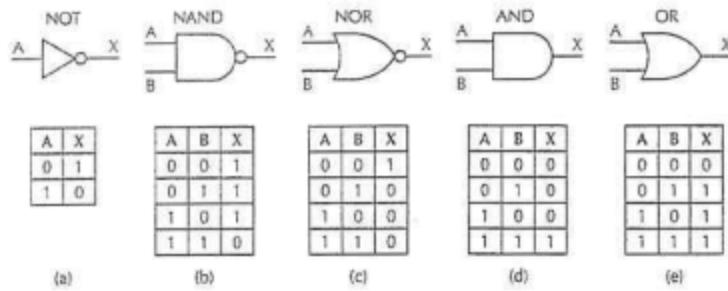
I circuiti digitali possono essere costruiti combinando tra loro un piccolo numero di componenti elementari.

Porte logiche

Un circuito digitale è un circuito in cui sono presenti due valori logici 0 e 1. La base hardware di tutti i calcolatori digitali è costituita da alcuni piccoli dispositivi elettronici, chiamati porte logiche, ciascuna delle quali calcola una diversa funzione di questi segnali.



I transistor hanno tre connessioni verso il mondo esterno: il collettore, la base e l'emettitore. Questi tre circuiti formano le tre porte logiche più semplici: NOT, NAND, NOR. Le parti NOT vengono anche chiamate **invertitori**.



Dalla seguente figura possiamo vedere i simboli usati per indicare alcune porte logiche e il loro comportamento funzionale.

Algebra di Boole

Per poter descrivere i circuiti combinando le porte logiche occorre definire un nuovo tipo di algebra in cui variabili e funzioni possono assumere soltanto i valori 0 e 1. Questa struttura viene chiamata **algebra di Boole**, ideata da George Boole. Una funzione booleana ha una o più variabili di input e genera un valore di output che dipende soltanto dal valore di queste variabili. Una funzione booleana di n variabili può essere completamente descritta mediante una tabella di 2^n righe. Questo tipo di tabella è nota come **tabella di verità**.

Implementazione delle funzioni booleane

Anche se abbiamo dimostrato la possibile implementazione di una qualsiasi funzione booleana mediante le porte NOT, AND e OR, spesso è più vantaggioso implementare circuiti utilizzando un solo tipo di porta logica. Sia le porte NAND che NOR forniscono un insieme di connettivi **funzionalmente completo**, nel senso che qualsiasi funzione booleana può essere calcolata usando soltanto uno di questi due tipi di porte.

Equivalenza di circuiti

Spesso i progettisti cercano di ridurre il numero di porte logiche utilizzate nei loro progetti per limitare le dimensioni delle schede con i circuiti stampati, diminuire il consumo energetico e aumentare la velocità. Bisognerà utilizzare l'algebra di boole per la ricerca di circuiti equivalenti, semplificare. Si AND che OR hanno proprietà algebriche che possono essere applicate per fare ciò. (CAP. 2 Libro ARM).

3.b Circuiti logici digitali elementari

Circuiti integrati

Le porte logiche non sono prodotte o vendute singolarmente, ma in unità chiamate **circuiti integrali**, cioè **IC** o **chip**. Un circuito integrale è un pezzo di silicio sul quale sono stampati i circuiti. I chip più piccoli utilizzano un supporto di tipo **DIP** (*Dual Inline Package*), due file di pin che vengono posizionate in un socket o alloggiamento. Due comuni supporti per i circuiti integrati più grandi sono **PGA** (*Pin Grid Arrays*) e **LGA** (*Land Grid Arrays*). I primi hanno pin sul fondo del supporto e si inseriscono in un alloggiamento mentre i secondi hanno piccoli pad piatti sul fondo del chip e non pin. I chip hanno un ritardo temporale finito, chiamato **ritardo della porta**, che comprende sia il tempo dovuto alla propagazione del segnale attraverso il chip, sia il tempo di comunicazione.

I chip possono essere divisi in classi in funzione del numero di porte logiche che contengono:

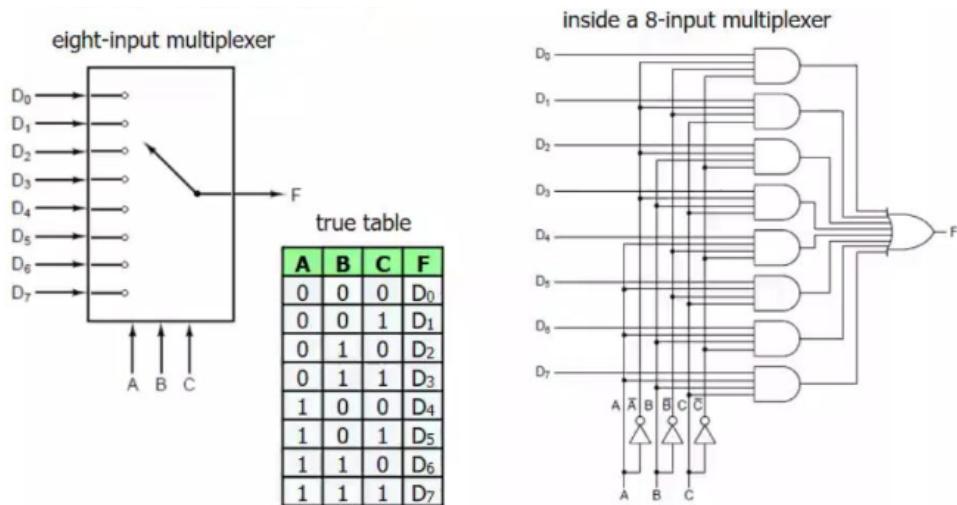
- **SSI** (*Small Scale Integrated*) sono circuiti con meno di 10 porte;
- **MSI** (*Medium Scale Integrated*) sono circuiti con 10-100 porte;
- **LSI** (*Large Scale Integrated*) sono circuiti con 100-100000 porte;
- **VLSI** (*Very Large Scale Integrated*) sono circuiti con più di 100000 porte;

Reti combinatorie

Molte applicazioni della logica digitale richiedono un circuito, chiamato **rete combinatoria**, con più input e output, in cui gli output sono unicamente determinati dagli input. Dove invece l'output dipende anche dagli ingressi di stato oltre che dagli input si chiama **circuito sequenziale**.

Multiplexer

Il **multiplexer** è un circuito con 2^n dati di input, un output e n input di controllo che permettono di selezionare uno dei dati di input, che viene instradato verso l'output. Il multiplexer può essere utilizzato per realizzare qualsiasi funzione logica.



Decoder

L'inverso del multiplexer è il **demultiplexer**, che redige il suo segnale di input verso uno dei 2^n output di base ai valori delle linee di controllo. È un circuito che riceve un numero a n-bit come ingresso e selezione in un'uscita l'unica linea corrispondente al suo valore numerico. È utile quando si dispone di un indirizzo e si vuole selezionare il chip di memoria corrispondente.

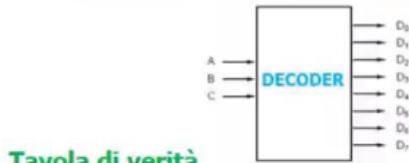
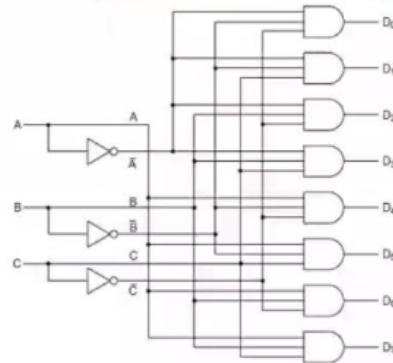


Tavola di verità

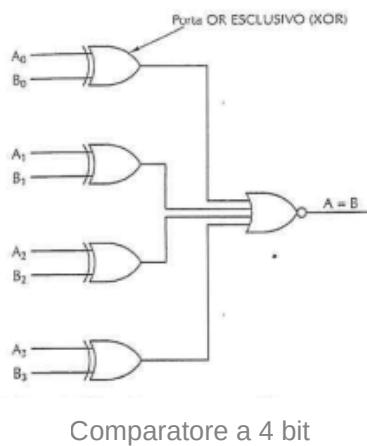
A	B	C	D ₀	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	D ₇
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	1

Schema digitale di un decoder a 3 ingressi



Comparator

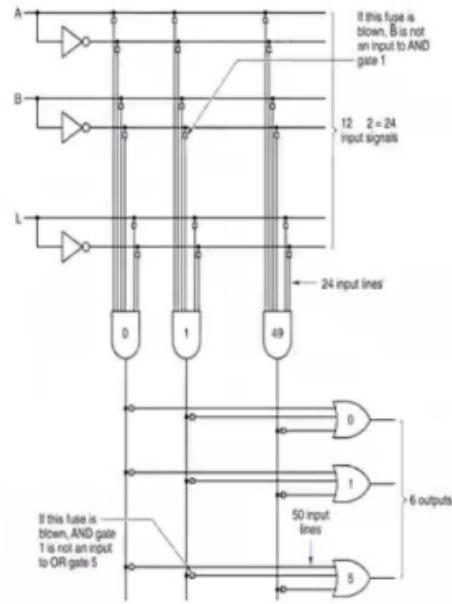
Il comparatore permette di confrontare due stringhe di bit, generando 1 se sono uguali sennò 0, si basa sulla porta logica XOR.



Comparatore a 4 bit

PLA (Programmable Logic Arrays)

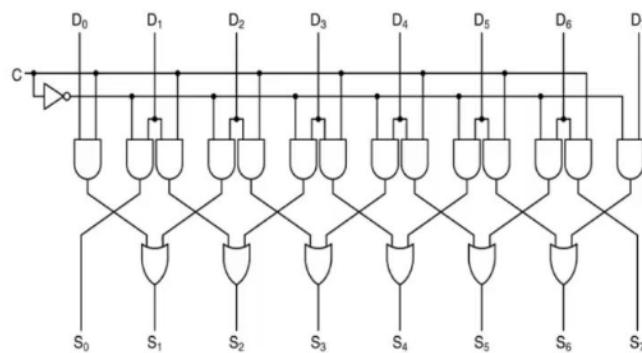
Un chip che permette di calcolare somme di prodotti è l'**array logico programmabile** o **PLA (Programmable Logic Arrays)**. Esempio di PLA con 12 ingressi: Il cuore del circuito è un array di 50 porte AND che creano una matrice 24*50. Ogni linea di ingresso delle porte AND contiene un fusibile



Circuiti per l'aritmetica

Shifer

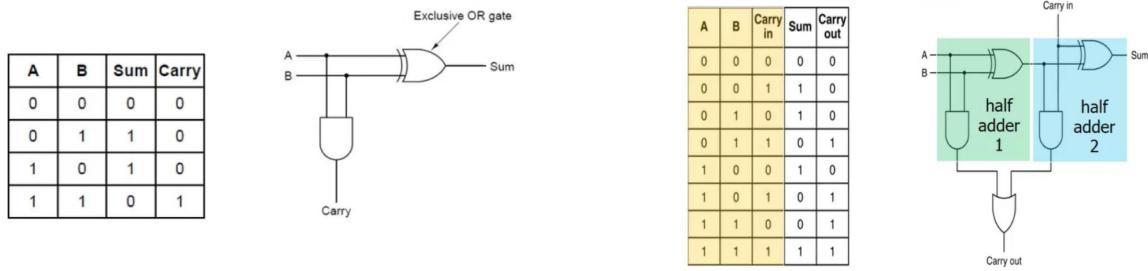
L'uscita è una parola che verrà fatta scorrere a destra o sinistra di un bit, il segnale C definisce la direzione dello scorrimento 0 a sinistra e 1 a destra inserendo un bit 0 nella posizione spostata.



Adder

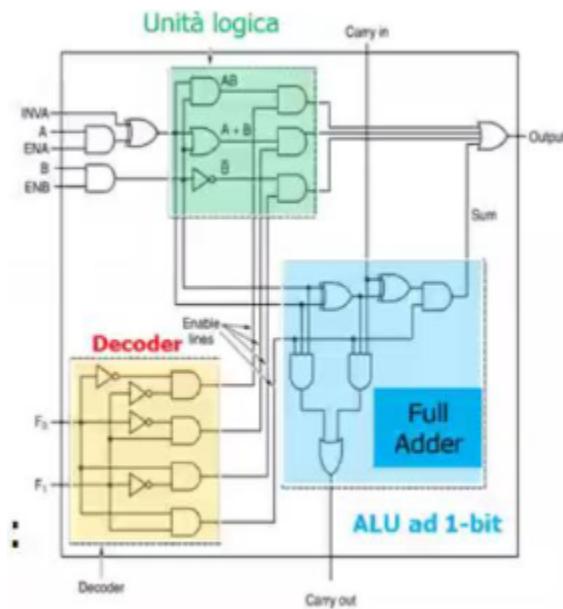
Il sommatore è un circuito in grado di eseguire la somma tra bit, considerando due variabili il circuito genera il risultato della somma e l'eventuale riporto. Il sommatore si può dividere in due categorie, **half-adder** e **full-adder**. Il primo caso non gestisce un riporto in ingresso restituendo in ingresso solo la somma. Il secondo caso è composto

da due half-adder e prende in ingresso 3 bit così da gestire anche il riporto in ingresso. Restituisce in ingresso somma e riporto

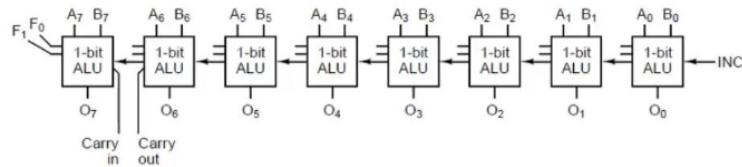


ALU

L'**ALU** o **unità aritmetico logica** contiene 3 differenti unità: un decoder, una unità logica e un full-adder. Il primo permette di selezionare l'operazione richiesta in base ai segnali in ingresso, la seconda è in grado di calcolare AB , $A + B$ e \bar{B} , la terza somma i due ingressi e il riporto in ingresso e calcola il risultato ed eventuale riporto.



Le ALU a 1-bit possono essere assemblate insieme per costruire ALU di lunghezza variabile. Questa tecnica è detta **bit slice** (suddivisione di bit) e può essere applicata anche ai precedenti circuiti digitali che lavorano bit a bit.



Clock

Il **clock** in questo contesto è un circuito che emette una serie di impulsi di larghezza definita e a intervalli di tempo costanti. L'intervallo temporale compreso tra impulsi consecutivi è detto **ciclo di clock** o **periodo**. Una tecnica che permette di aumentare la risoluzione del segnale di clock è di effettuare un AND tra il segnale originario e una sua replica ritardata.

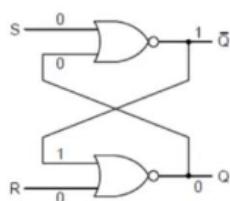
3.c Memoria

Un componente essenziale di ogni calcolatore è la memoria, perché utilizzata per memorizzare i dati e le istruzioni. Per costruire un circuito che memorizza dati è necessario utilizzare i circuiti sequenziali in cui l'uscita dipende non solo dagli ingressi ma anche dallo stato del circuito. Un circuito sequenziale quindi è in grado di mantenere dei dati, quando è alimentato.

Latch

Un circuito sequenziale semplice può essere realizzato con due porte logiche NAND o NOR che hanno uscite retroazionate in ingresso (che dipendono dai valori dei precedenti ingressi). Per creare una memoria a 1 bit è necessario disporre di un circuito che in qualche modo ricordi i precedenti input. Il Latch Set-Reset **SR Latch** ha due ingressi: S è utilizzato per impostare ad 1 il valore dell'uscita e R usato per azzerarlo. Supponiamo che S e R siano 0, abbiamo due possibili stati.

a) $Q=0$ e $\bar{Q}=1$



b) $Q=1$ and $\bar{Q}=0$

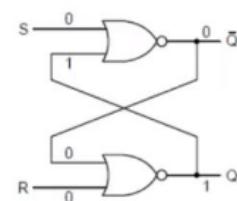


Tavola di verità

A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

Questi sono condizioni di stabilità quando gli ingressi non cambiano (e il circuito è alimentato).

Se S diventa 1 mentre Q era 0 esso imposta Q a 1.

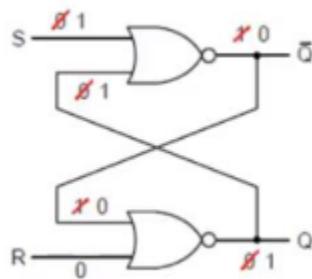
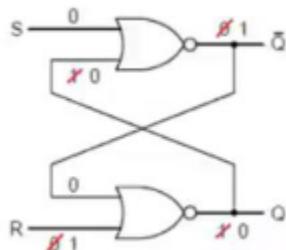


Tavola di verità del NOR

S	Q	NOR
0	0	1
0	1	0
1	0	0
1	1	0

R	Q̄	NOR
0	0	1
0	1	0
1	0	0
1	1	0

A questo punto se $Q = 1$, S non potrà più cambiare lo stato del circuito che si trova in una condizione di stabilità. Supponiamo che R diventi 1 (Q è ad 1), esso resetta lo stato del latch poiché Q diventa 0:



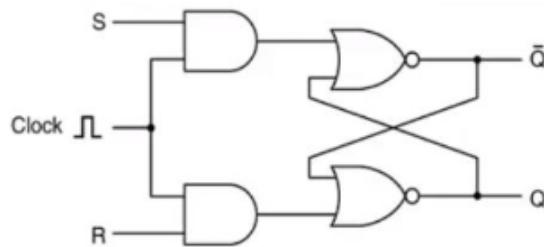
R	Q̄	NOR
0	0	1
0	1	0
1	0	0
1	1	0
1	0	0

R	Q̄	NOR
0	0	1
0	1	0
1	0	0
1	1	0
1	0	0

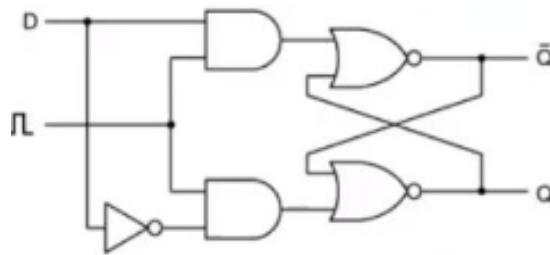
Non sono possibili altri cambiamenti di stato se cambia R quando $Q = 0$.

Spesso è preferibile impedire che un latch cambi di stato se non in specifici momenti.

L'SR Latch clocked è il latch in cui i segnali S ed R agiscono solo quando il segnale di clock è attivo. Può essere realizzato semplicemente aggiungendo al Latch SR due porte AND che mascherano i segnali originali S ed R.



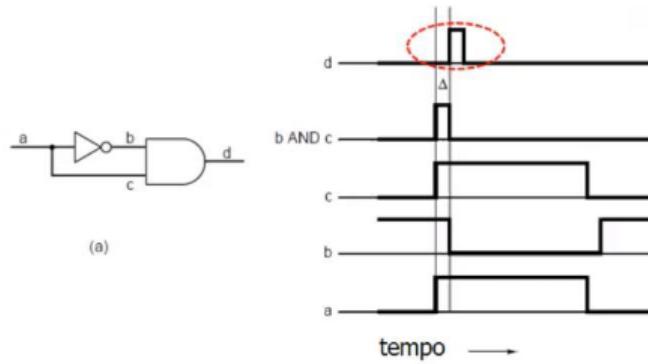
Il latch SR hanno uno stato instabile quando $S=R=1$, in questa circostanza il comportamento del circuito non è deterministico. Per ristabilire la condizione di equilibrio occorre impostare i valori $S=R=0$. Il latch SR è il primo circuito di memoria che ha la capacità di ricordare in Q e l'ultimo valore impostato per S o R. Per risolvere l'ambiguità del latch SR (causata dalla condizione $S=R=1$) può essere utilizzato un solo ingresso dati (D). Quando $D=1$ e il clock è alto, il latch va nello stato $Q=1$, quando $D=0$ e il clock è alto, il latch va nello stato $Q=0$.



Questo circuito è detto **D latch clocked** e realizza una cella di memoria ad 1-bit dove il valore memorizzato è sempre disponibile sulla linea Q. Per caricare in memoria il valore corrente di D occorre spedire un impulso positivo sulla linea del clock.

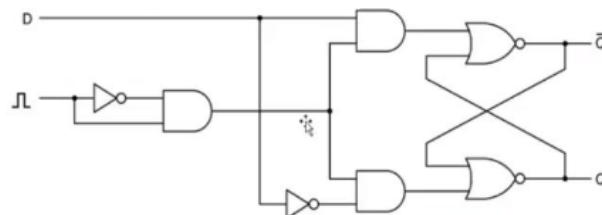
Flip-flop

In molti circuiti è necessario campionare il valore di una certa linea in un particolare istante e memorizzarlo. In questi circuiti, chiamati flip-flop, la transizione di stato non si verifica quando il clock vale 1, ma durante la transizione del segnali da 0 a 1 oppure da 1 a 0. A causa di questo comportamento il flip-flop si dice **edge triggered**. In questa situazione la lunghezza dell'impulso del clock non ha alcuna importanza. Un flip-flop è a **commutazione sul fronte**, mentre un latch è a **commutazione a livello**. Un generatore di impulsi si può realizzare semplicemente utilizzando una porta logica AND e un inverter che ritarda il segnale in uno dei due ingressi così in figura:



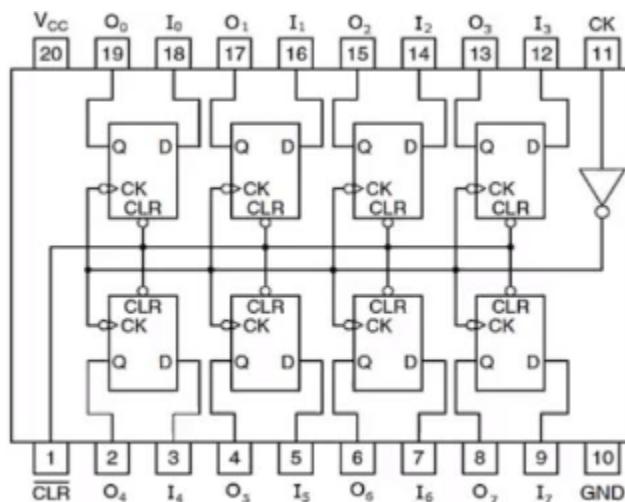
Si osservi anche il segnale d subisca il ritardo della porta AND.

Il generatore di impulsi sostituisce il segnale di clock e permette di campionare il valore dell'ingresso D in un intervallo di tempo ridotto. Molti latch hanno due ingressi aggiuntivi Preset (per forzare lo stato Q=1) e Clear (per forzare lo stato Q=0).



Registri

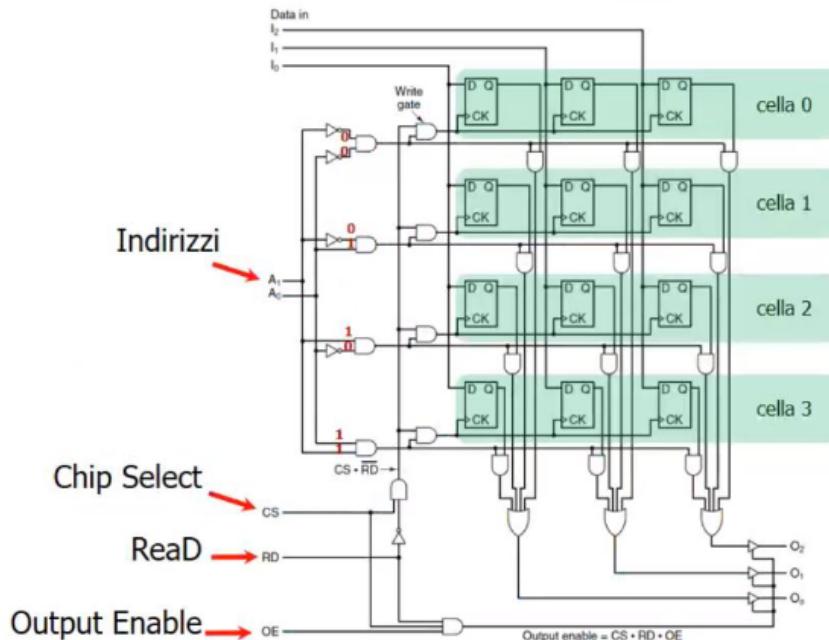
I flip-flop combinati creano registri che memorizzano dati composti da più bit. Per realizzare un registro ad 1-byte possono essere utilizzati 8 flip-flop D.



Organizzazione della memoria

Memorie di grandi dimensioni utilizzano un indirizzo per localizzare la cella sulla quale eseguire una operazione di lettura/scrittura, ogni cella ha lo stesso numero di bit per memorizzare i dati. Il circuito usa due bit per indirizzare le celle (A), tre bit per l'ingresso della parola (I), tre bit per l'uscita della parola (O) e tre segnali di controllo:

- **CS (Chip Select)** usato per selezionare il circuito.
- **RD (ReaD)** utilizzato per specificare se scrittura (0) o lettura (1).
- **OE (Output Enable)** utilizzato per abilitare le uscite.
- La dimensione di una memoria generica con n linee di indirizzamento che memorizza m bit di informazione per ogni cella è pari a: $2^n * m$ bits.

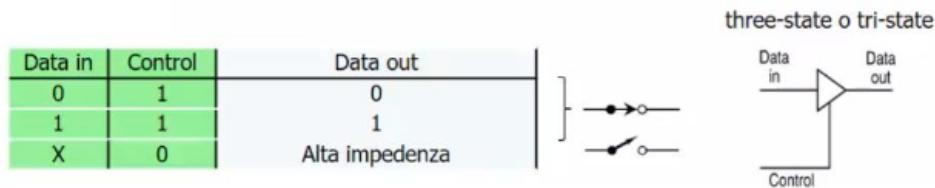


Mentre si possono collegare insieme più ingressi, quando si collegano due o più uscite c'è il potenziale rischio del corto circuito. Nelle memorie attuali gli ingressi e le uscite occupano le stesse linee, quindi si deve far in modo di scollegare le uscite nelle operazioni di scrittura. Per risolvere questo problema esistono due differenti soluzioni: **open collector** e **buffer three state**.

Il primo caso il segnale in uscita alla porta logica è applicato alla base di un transistor che ha il collettore aperto in uscita, se l'output è H il collettore è un circuito chiuso

l'emettitore, se l'output è L il collettore è aperto. Questo permette di collegare più uscite insieme in uno schema chiamato OR-cablato o wired-OR.

Nel buffer a tre stati invece l'uscita assume un terzo stato di alta impedenza (come se fosse un circuito aperto), la porta logica si comporta come un interruttore che abilita o meno il passaggio del segnale dall'ingresso all'uscita.



Chip di memoria

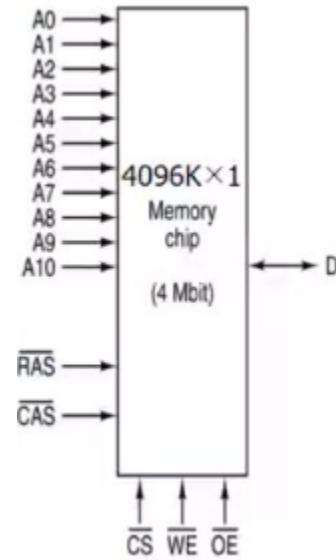
I chip di memoria hanno schemi riusabili facilmente espandibili. Alcuni segnali sono in logica positiva altri in logica negativa, se sono attivi a livello alto o basso. Per evitare confusione si dice segnale asserito se attivo e negato se non lo è.

Poiché un computer ha normalmente molti chip di memoria sono necessari alcuni segnali fondamentali:

- **CS (Chip Select)** usato per selezionare il chip da attivare.
- **RD (ReaD)** utilizzato per specificare se scrittura (0) o lettura (1).
- **OE (Output Enable)** consente di fondare insieme i segnali di uscita.

Nella figura è rappresentata una memoria da 4Mbit cioè $2^{11} * 2^{11}$ celle da un bit per leggere o scrivere occorre:

- Selezionare una riga in A e asserire il segnale RAS
- Selezionare la colonna in A e asserire il segnale CAS
- Leggere/scrivere D



Questa organizzazione riduce il numero di pin ma rende il chip più lento rispetto ad uno equivalente con indirizzamento lineare (che richiederebbe un solo ciclo per la selezione della cella ma 11 pin in più per l'indirizzamento).

RAM e ROM

La **RAM** (*Random Access Memory*) sono memorie che hanno lo stesso tempo di accesso indipendentemente dalla posizione della cella, esistono due tipi di RAM:

- RAM statiche (**SRAM**) costruite utilizzando circuiti simili ai flip-flop D che sono in grado di mantenere l'informazione fintanto che sono alimentate. Sono molto veloci e utilizzate per le cache di secondo livello.
- RAM dinamiche (**DRAM**) non usano flip-flop ma usano un array di celle con un transistor ed un piccolo condensatore che può essere caricato (1) o scaricato (0) ma deve essere rinfrescato. Le DRAM hanno maggior capacità delle SRAM ma richiedono circuiti più complessi.

Le **ROM** (*Read-Only Memory*) sono memorie a sola lettura il cui contenuto non cambia anche quando non sono alimentate. I dati in una ROM sono inseriti in fase di costruzione.

La **PROM** (*Programmable ROM*) è una ROM che può essere programmata una sola volta bruciando dei fusibili collocati nelle intersezioni della matrice

La **EPROM** (*Erasable PROM*) funziona come una PROM che può essere

riprogrammata attraverso l'esposizione ai raggi ultravioletti per 15 min.

La **EEPROM** (*Electrically EPROM*) funziona come una EEPROM che per essere cancellata (byte a byte) non deve essere inserita in una camera speciale, per l'esposizione alla luce ultravioletta, ma è sufficiente applicare un impulso elettrico.

La **memoria flash** è un tipo speciale di EEPROM che può essere cancellata a blocchi (e non a byte).

3.d Chip della CPU e bus

Chip della CPU

Tutte le CPU moderne sono contenute in un unico chip che possiede un insieme di pin (e segnali) attraverso i quali la CPU comunica con il mondo esterno. I pin/segnali della CPU possono essere divisi in tre categorie: **indirizzi, dati e controllo**. I pin sono connessi agli altri componenti attraverso il **bus**.

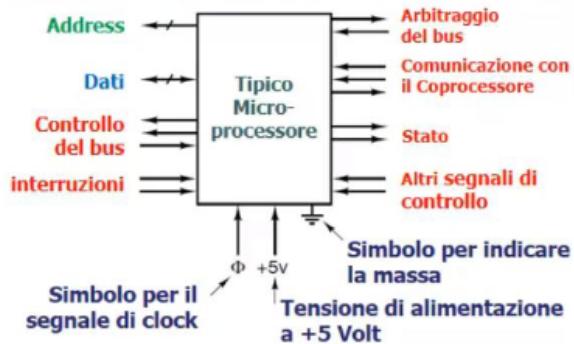
Quando la CPU esegue un'istruzione (l'intero processo viene ripetuto per ogni parola di dati):

- Inizialmente pone l'indirizzo di memoria dove si trova l'istruzione in memoria sul bus indirizzi;
- Informa la memoria che vuole eseguire una operazione di lettura (utilizzando le linee di controllo del bus);
- La memoria risponde ponendo la parola sul bus dati e asserendo un segnale che indica che l'operazione è stata svolta;
- La CPU riceve questo segnale di controllo, legge i dati e si predisponde per eseguire l'istruzione.

I principali parametri che determinano le prestazioni di una CPU sono il numero di pin per l'indirizzamento e per i dati. Una CPU con m pin di address può indirizzare fino a 2^m locazioni di memoria, con n pin dati può leggere/scrivere una parola di n -bit in una singola operazione. I pin di controllo possono essere raggruppati nelle seguenti categorie:

- Interruzioni
- Arbitraggio del bus

- Controllo del bus
- Comunicazione con il coprocessore
- Stato
- Vari altri segnali di controllo



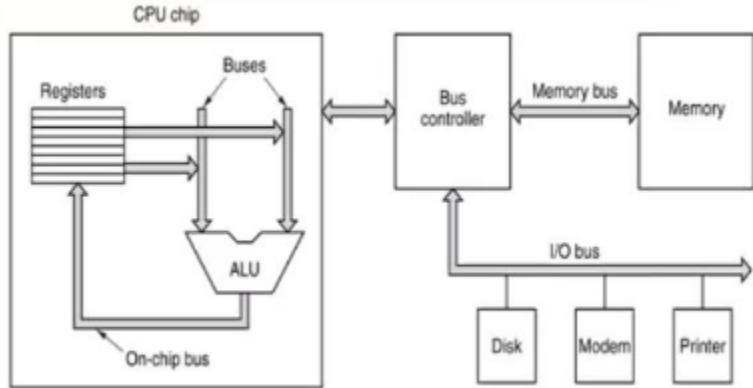
I pin di interrupt sono degli input che giungono alla CPU dalle periferiche. Nella maggior parte dei sistemi la CPU può comunicare a un dispositivo I/O l'inizio di un'operazione mentre questo porta avanti il proprio lavoro, passa a eseguire qualche altra operazione. Non appena il dispositivo ha completato la propria operazione asserisce un segnale su uno di questi pin in modo da interrompere la CPU e permetterle di utilizzare il dispositivo, per esempio per verificare se si siano verificati errori di I/O. Alcune CPU hanno un pin di output con il quale possono anche spedire una conferma in risposta di un segnale di interrupt.

I pin di arbitraggio del bus sono necessari per regolare il traffico sul bus, allo scopo di evitare che due dispositivi cerchino di usarlo nello stesso momento. Anche la CPU fa richiesta di utilizzo del bus.

Alcune CPU sono progettate per funzionare con coprocessori come i chip di virgola mobile oppure, i chip grafici o di altro tipo. Per facilitare la comunicazione tra CPU e coprocessore ci sono dei pin speciali per ricevere e soddisfare vari tipi di richieste. Oltre a questi segnali alcune CPU possono o per resettare il calcolatore o effettuare operazioni di debugging, avere altri pin per altri scopi, per esempio ricevere informazioni di stato, o garantire compatibilità con chip di I/O più vecchio.

Bus del calcolatore

Un **bus** è un collegamento elettrico comune che unisce vari dispositivi. I bus possono essere **interni** alla CPU (connessione registri-ALU) oppure **esterni** per connetterla alla memoria o alle periferiche I/O. Mentre i progettisti di CPU possono utilizzare il tipo di bus che desiderano, per i bus esterni devono essere definiti delle regole (**protocollo del bus**) da rispettare. I primi PC avevano un solo bus di sistema, oggi esistono più bus per il collegamento **CPU-memoria** e **CPU-device di I/O**.



Alcuni dispositivi nel calcolatore sono **attivi** (o **master**) perché possono iniziare un trasferimento dati sul bus, altri **passivi** (o **slave**) poiché restano in attesa di una richiesta di un master.

Per amplificare il segnale dei master sul bus è utilizzato un chip chiamato **bus driver**, viceversa gli slave usufruiscono di un chip chiamato **bus receiver** per il collegamento con il bus. Per le periferiche che possono svolgere sia il ruolo di master sia quello di slave si utilizza un chip chiamato **driver-receiver bus**.

Questi chip di interfacciamento per evitare il problema di connettere più uscite insieme utilizzano delle porte logiche **buffer tri-state** oppure il collegamento in **wired-OR** di più porte open collector.

Aampiezza del Bus

Se un bus ha n linee di address, allora la CPU può indirizzare almeno 2^n diverse locazioni di memoria. L'incremento della dimensione del bus indirizzi (in termini di bit) permette di indirizzare maggiori spazi di indirizzamento ma incrementa anche i costi, i fili e connettori.



Si nota come l'incremento delle linee e la retrocompatibilità sono due caratteristiche contrapposte (il primo bus da 20 bit resta così come il 2 bus da 4 bit resta nella 3 CPU). Per incrementare la larghezza di banda del bus si può:

1. Ridurre il periodo del clock del bus (più trasferimenti al secondo) è possibile ma difficile perché si corre il rischio di:
Avere più dispositivi che operano a velocità differenti (**problema del disallineamento del bus**) e perdere la retrocompatibilità.
2. Incrementare **l'ampiezza del bus dati** (più bit nell'unità di tempo). Per non avere bus troppo ampi (ed effetti collaterali) si può utilizzare una tecnica di **multiplexing** delle linee sia per dati sia per indirizzi, ovviamente questo conduce ad un rallentamento del sistema.

Temporizzazione del bus

Dal punto di vista della temporizzazione esistono differenti approcci:

Bus sincroni

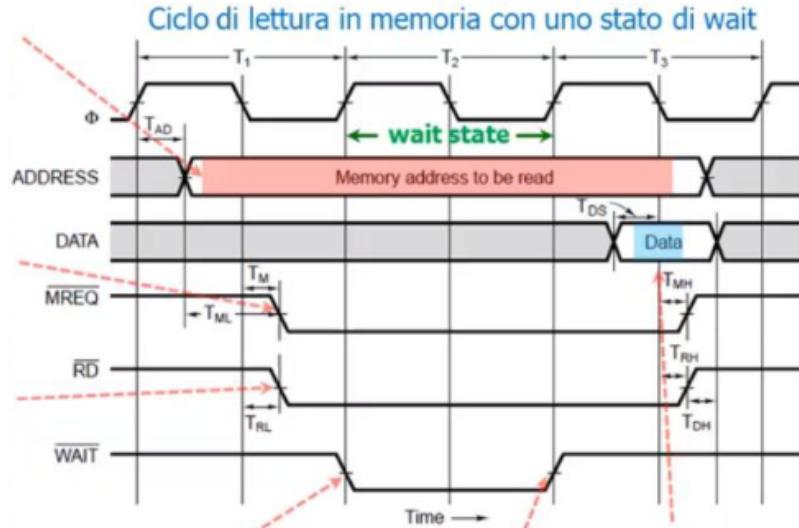
Utilizzano un clock che determina la temporizzazione delle attività sul bus (**ciclo di bus**): ogni operazione richiede un numero di periodi di clock per potere essere eseguita.

Per questioni di affidabilità e retrocompatibilità i bus moderni operano a velocità di clock non elevata.

1. La CPU (master) pone l'indirizzo di

memoria sull'address bus in modo che le linee si stabilizzino.

2. La CPU comunica al sistema che l'operazione che intende svolgere è con la memoria.
3. La CPU comunica che si tratta di una operazione in lettura, a questo punto la memoria (slave) deve fornire sul data bus il contenuto della cella indirizzata dall'address bus.

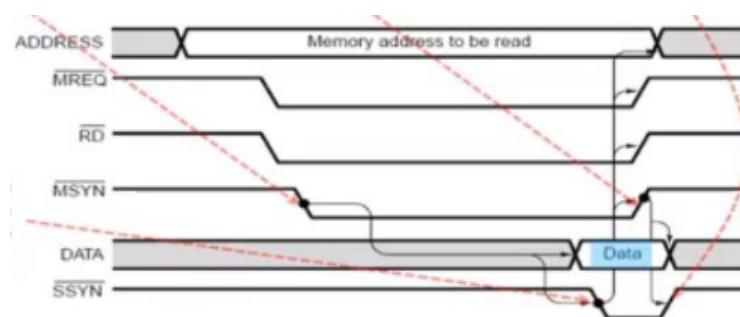


1. (4) Poiché la memoria è meno veloce della CPU, le chiede uno stato di attesa aggiuntivo (wait state).
2. (5) La memoria è pronta a fornire i dati allora nega il segnale di WAIT e la CPU potrà leggere i dati.
3. (6) La CPU legge i dati sul fronte in discesa

Bus asincroni

Il ciclo di bus può avere qualsiasi durata e quindi è più efficiente. Tuttavia è più difficile da costruire rispetto ad uno sincrono.

1. Il segnale MSYN è utilizzato dal master per richiedere l'inizio di una operazione.
2. SSYN è usato dallo slave per avvisare che i dati sono disponibili.



I 4 eventi in figura mostrano un **full handshake** tra master e slave, il tutto avviene senza base dei tempi. In questo modo il bus si adatta alla velocità del dispositivo collegato ed un dispositivo lento non rallenta l'intero sistema. La maggior parte dei bus sono sincroni perché più semplice da realizzare a causa degli enormi investimenti fatti fin ora.

Arbitraggio del bus

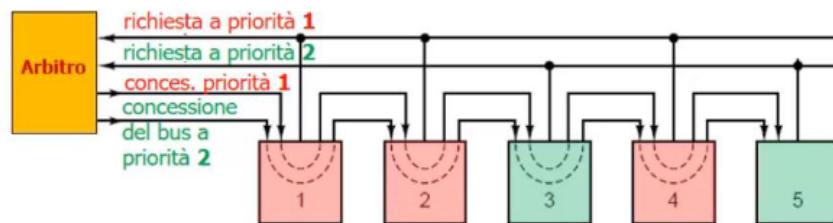
Ciascun dispositivo “intelligente” del computer può diventare, a turno, master del bus. L’arbitraggio del bus è utilizzato per prevenire situazioni di conflitto in cui due o più dispositivi tentano di diventare, nello stesso momento, master del bus. L’arbitraggio può essere **centralizzato** o **decentralizzato**.

Il primo necessita della presenza di un **arbitro**, normalmente integrato nel chip della CPU, che può diventare anche un fattore di criticità in caso di rottura. Quando l’arbitro riceve una richiesta, concede la richiesta asserendo una linea di concessione del bus. Quando il dispositivo più vicino dell’arbitro vede la concessione:

- Se è lui che lo ha chiesto, blocca la linea negandola a tutti i suoi successori
- Altrimenti mantiene asserita la linea.

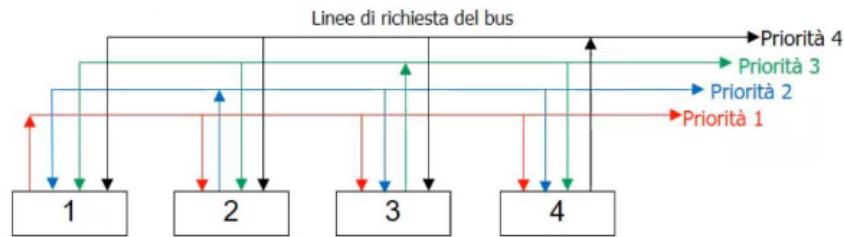
Quando due, o più, dispositivi fanno richiesta vince il più vicino all’arbitro. Questo schema è chiamato **collegamento a festone** (*daisy chaining*) ed ha la proprietà di assegnare ai dispositivi diverse proprietà in base alla vicinanza con l’arbitro.

Per superare il problema che la priorità è cablata nella connessione molti bus definiscono più livelli di priorità utilizzando differenti linee richiesta-concessione (4,8 o 16).



Nel arbitraggio decentralizzato Ogni dispositivo ha una propria linea di richiesta ed una priorità, prima di richiedere il bus ciascuno deve verificare che non ci sia già una richiesta con priorità più alta. Al termine dell’utilizzo del bus, la linea di richiesta deve

essere negata. Gli svantaggi è che ci sono troppi collegamenti, il numero di dispositivi non può superare il numero di linee.



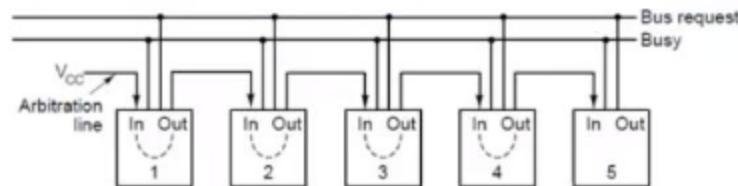
Un differente schema decentralizzato di arbitraggio utilizza tre linee:

- La linea di richiesta del bus (in configurazione wired-OR).
- La linea the busy asserita dal dispositivo bus master corrente.
- La linea di arbitraggio propagata tra i dispositivi in cascata.

Per ottenere il bus un dispositivo deve:

- Controllare che busy sia negata e che l'ingresso IN sia asserito.
- In questo caso nega OUT, asserisce la linea busy e diventa master del bus.

Al termine, sblocca OUT asserendolo e libera busy negandolo.



Operazioni del bus

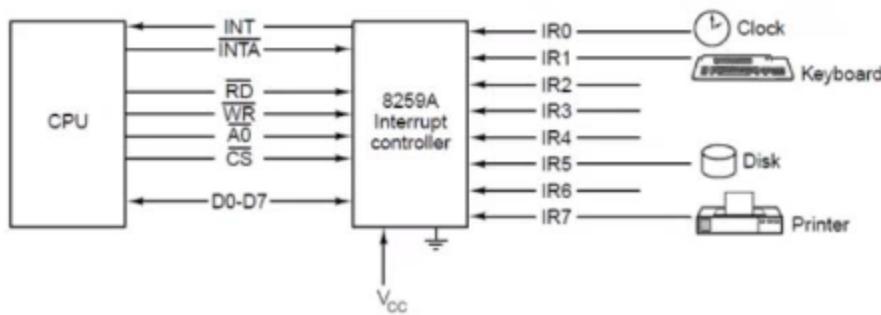
Di solito viene trasferita una parola alla volta. Tuttavia quando si utilizza la cache è preferibile prelevare in una volta sola un'intera linea di cache. Spesso i trasferimenti a blocchi possono essere effettuati in modo più efficiente rispetto a una sequenza di trasferimenti individuali. Il master del bus comunica quante parole devono essere trasferite inserendo il conteggio sulle linee dati, lo slave, invece di restituire un'unica parola, genera un output a ogni ciclo di parola finché il contatore non sia asserito.

Esistono anche altri tipi di cicli di bus, per esempio su un sistema multiprocessore con due o più CPU sullo stesso bus è spesso necessario assicurarsi che in un dato

momento soltanto una CPU utilizzi delle strutture dati critiche della memoria. Si imposta 1 o 0 alla struttura dati quando è in uso o no. Se una CPU vuole acquisire l'accesso deve leggere il valore e impostarla a 0 o 1. Il problema è che due CPU potrebbero leggere la variabile durante cicli di bus consecutivi e se entrambe vedono che la variabile è 0 la imposteranno a 1 e penseranno di essere l'unica CPU d utilizzare la struttura dati. Per evitare ciò questi sistemi multiprocessori hanno spesso uno speciali ciclo di bus che consente a una qualsiasi CPU di azionare sulla struttura dati senza rilasciare il bus evitando che altre CPU se ne impadroniscono interferendo con l'operazione della prima CPU.

Un altro importante tipo di ciclo di bus serve a gestire gli interrupt. Quando la CPU ordina a un dispositivo di I/O di effettuare qualche operazione si aspetta di solito un interrupt alla fine del lavoro; la segnalazione di questi interrupt richiede l'utilizzo del bus. Quando uno o più dispositivi richiedono un'interruzione si asserisce il segnale INT (Interrupt) della CPU. Se la CPU è in grado di gestire la richiesta di interruzione, risponde al controllore con il segnale INTA (INT Acknowledge), a questo punto deve porre sul data bus il numero del dispositivo che ha generato l'interrupt.

La CPU utilizza quel numero come indice per accedere al vettore di interruzione e trovare l'indirizzo della ISR (Interrupt Service Routine).



3.e Esempi di chip della CPU

Pentium 4

Il Pentium 4 è un diretto discendente della CPU 8088 utilizzata nei primi PC IBM. Contiene 55 milioni di transistor (la larghezza di linea cioè la distanza tra transistor è di 90 nm), ed opera a una frequenza di clock fino a 3,2 GHz. È in grado di scambiare dati con la memoria a 64-bit ma dal punto di vista software il programmatore vede una

macchina a 32-bit (compatibile con i vecchi software). La microarchitettura interna NetBrust è molto diversa dai suoi predecessori:

- Ha una pipeline più profonda delle precedenti architetture.
- Due unità ALU che operano al doppio della frequenza di clock.
- Supporta l'hyperthreading.
- Ha due insiemi di registri che permettono ai programmi di funzionare come se ci fossero due CPU fisiche.

A seconda del modello può avere 2 o 3 livelli di cache. Tutti i modelli hanno 8KB di SRAM sul chip per la cache di primo livello. Il secondo livello di cache è in grado di memorizzare fino a 1 MB. L'Extreme Edition ha anche 2 MB per la cache di terzo livello. Ogni CPU mantiene la consistenza tra le cache attraverso il processo di **snooping** dei riferimenti di memoria sull'address bus.

Ha due bus sincroni primari, il **memory bus** è utilizzato per accedere alla memoria principale DRAM, il **bus PCI** è utilizzato per il colloquio con i dispositivi di I/O.

Un problema comune a tutte le attuali CPU è il consumo energetico ed il calore prodotto. In accordo con le leggi della fisica qualsiasi dispositivo elettronico che produce molto calore deve assorbire molta energia, in un computer portatile utilizzare troppa energia non è desiderabile poiché la batteria si consuma presto, perciò fu progettato un modo per addormentare la CPU quando è in uno stato di inattività e in un sonno profondo.

Sono stati definiti 5 stati di funzionamento attivo al sonno profondo. Negli stati intermedi alcune funzionalità importanti (come lo snooping della cache e la gestione degli interrupt handling) sono abilitate, mentre altre sono disattivate. Quando la CPU è in sonno profondo:

- I valori delle cache e dei registri sono preservati, ma il clock e le unità interne sono spente.
- Solo un segnale hardware può risvegliarla.

Ha 478 pin: 198 per i segnali, 85 di alimentazione, 180 per la massa e 15 liberi per usi futuri.

Segnali di **arbitraggio del bus**: *BR0#* per la richiesta del bus, *BPRI#* per le richieste del bus ad alta priorità e *LOCK#* per indicare che il bus è occupato.

Segnali **dell'address bus e di controllo**: *A#* 36 bit di indirizzamento di cui 3 bit meno

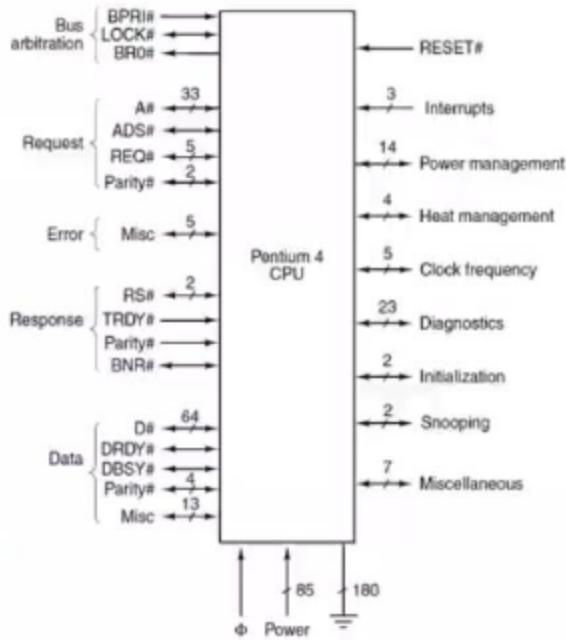
significativi impostati a 0, *ADS#* è utilizzato per indicare che il bus indirizzi contiene un riferimento valido e *REQ#* specifica il tipo di operazione (lettura/scrittura).

Ci sono vari segnali di errore per rappresentare errori derivanti dal calcolo, dei dispositivi interni dei controlli macchina e altri di tipo generico.

Segnali di **risposta** relativi al bus: *RS#* contiene il codice di stato, *TRDY#* indica che lo slave è pronto per accettare i dati e *BNR#* permette allo slave di inserire stati di attesa.

Segnali del **bus dati**: *D#* sono i 64 bit del bus dati, *BNR#* indica che il bus è occupato e *DRDY#* indica che i dati sono pronti sul bus.

Nel P4 si possono utilizzare le interruzioni come nell'8080 oppure attraverso il controller di interruzione programmabile avanzato **AICP** (Advanced Programmable Interrupt Controller).



Le CPU attuali sono più veloci delle memorie centrali quindi è essenziale ottenere il massimo throughput dalla memoria per non lasciare in attesa la CPU. Il bus che collega la CPU alla memoria è altamente parallelizzato, con 8 transizioni correnti.

Intel Core i7

è una macchina a 64 bit, anch'esso è un diretto discendente della CPU 8088. La prima versione rilasciata nel 2008 conteneva 4 core con 731 milioni di transistor. Ogni core è hyperthreaded, più thread hardware attivi in parallelo sullo stesso core. Dispone di 3

livelli di cache, ciascun core effettua uno snooping sul bus di collegamento con la memoria per garantire la consistenza delle informazioni.

Il primo livello di cache ha due distinte cache L1 da 32 KB per dati e istruzioni, per core.

Il secondo livello ha una cache da 256KB integrata di dati e istruzioni, per core.

Il terzo livello ha una cache da 4 a 15MB condivisa tra i core.

UltraSPARC III

La famiglia UltraSPARC era la linea di CPU RISC a 64-bit prodotta dalla Oracle. Era pienamente compatibile con la precedente architettura a 32-bit e veniva utilizzata per le workstation e i server prodotti dalla Oracle. Il chip era prodotto dalla Texas Instruments. Nel 2002, la larghezza di linea era di 130 nm e il clock di 1,2 GHz. Questi chips consumavano 50 W di potenza e avevano gli stessi problemi di dissipazione termica del P4.

Ha 2 cache di L1: 32KB dedicati alle istruzioni e 64KB per i dati, 2 cache di L2: 2KB per il prefetch e 2 KB usata per collezionare le scritture. Il controller della cache e la logica di ricerca dei blocchi di cache era all'interno del chip, invece la memoria era esterna. Questo permise ai progettisti di scegliere la memoria con il miglior rapporto qualità prezzo senza essere vincolati dalla CPU. L'UltraSPARC III utilizzava un address bus di 43-bit che le permetteva di gestire fino 8 TB di memoria principale. Il bus dati era di 128-bits quindi era in grado di trasferire 16 bytes per volta. La velocità del bus era di 150MHz, questo permetteva una banda di 2.4GB/sec nel collegamento con la memoria, molto più rispetto ai 528MB/sec dell'attuale bus PCI.

Per collegare più CPU UltraSPARC con più memorie, Oracle sviluppò l'architettura UPA. che può essere realizzato come un bus o uno switch.

Tutta la memoria principale è divisa in linee di cache dette blocchi da 64 byte. La cache L1 contiene: 256 blocchi di istruzioni utilizzate più pesantemente e 25 blocchi di dati più pesantemente utilizzate. I blocchi che non entrano nella L1 sono memorizzati nella cache L2

CPU economiche

P4 e Ultra sono stati esempi di CPU ad alte prestazioni per la costruzione di PC e server estremamente veloci. Esiste un'altra classe di CPU destinate ai sistemi embedded che possiamo trovare dentro elettrodomestici, protesi etc.. Il costo di questi computer è estremamente ridotto e può arrivare persino ad 1 euro. I computer

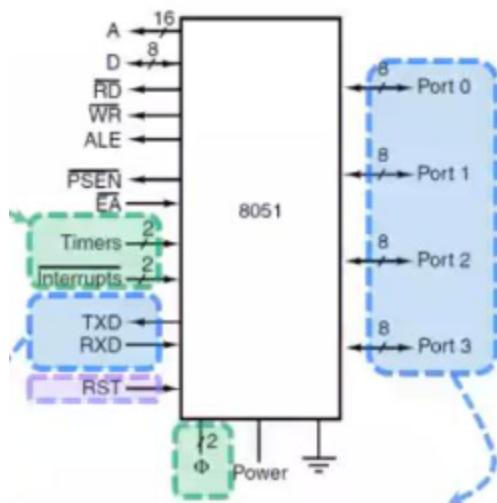
all'interno di questi apparati tendono ad essere economici piuttosto che ad avere prestazioni elevate.

Intel 8051

I chip Intel 8051 è stato uno tra i più diffusi microcontrollori nelle applicazioni di controllo industriali in virtù del suo basso costo. È un circuito integrato da 40 pin con 16-bit di address e 8-bit per il bus dati. A differenza di una CPU pura ha 32 linee di I/O, organizzate in 4 gruppi di 8 bit ciascuno. Ha 4 set di registri ciascuno da 8 registri e l'ALU sono a 8-bit. A è il bus indirizzo a 16-bit per la memoria esterna, D contiene un bus dati a 8-bit. I segnali RD e WR servono a leggere o scrivere dati da/verso la memoria esterna.

- ALE (Address Latch Enable) indica la presenza di un indirizzo valido sul bus.
- PSEN (Program Store Enable) indica che la CPU vuole leggere il programma dalla memoria.
- EA (External Access) può essere collegato:
 - alto, per usare sia la memoria interna sia per quella esterna.
 - low, per utilizzare soltanto quella esterna.

Ha due clock esterni, due contatori a 16-bit, due livelli di priorità di interruzione, asseriti al livello basso. Le I/O sono: TXD, per l'uscita seriale, RXD, per l'ingresso seriale e 4 porte bidirezionali ciascuna con 8 bit paralleli. Il reset del chip avviene con il segnale RST.



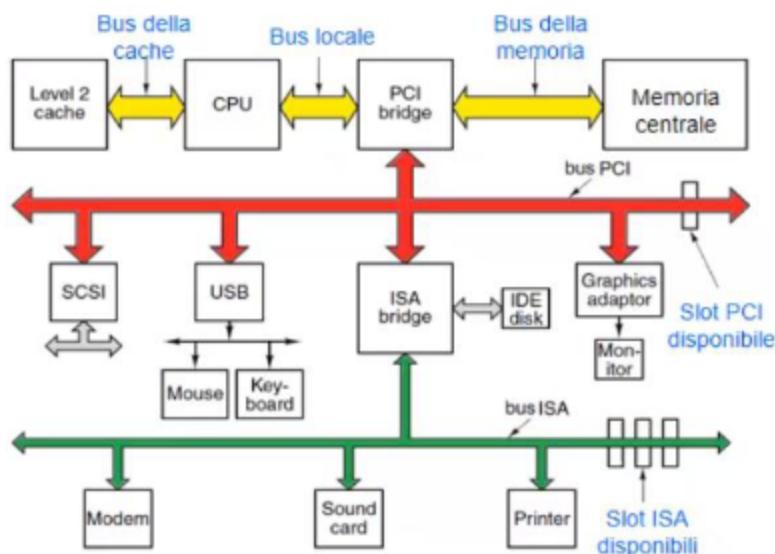
3.f Esempi di bus

I bus sono il collante che tiene insieme le componenti del computer. Oggi i più diffusi sono il bus PCI, PCIe e l'USB. Mentre l'USB è un bus per periferiche a bassa velocità, il PCI e PCIe sono utilizzati per connettere le periferiche veloci. Il bus del primo PC IBM è stato lo standard de facto dei sistemi basati su architettura 8088 era parallelo e aveva 62 segnali tra cui: Segnali di controllo, 20 bit per l'address bus, 8 bit per il bus dati e Altri segnali. Per mantenere la compatibilità con le schede esistenti quando nacque il PC/AT 80286 fu aggiunto un secondo connettore.

Il bus ISA (Industry Standard Architecture) era una copia di quello del 80286 e funziona con un clock di 8,33 MHz. Il successore del bus ISA fu esteso a 32-bit e, per questa ragione, fu chiamato EISA (Extended ISA).

PCI

Con l'introduzione dei giochi multimediali e video a pieno schermo, la velocità offerta dal bus ISA divenne presto insufficiente. Il bus PCI (Peripherical Component Interconnect) può funzionare con una frequenza di clock fino a 66 MHz, gestire trasferimenti a 64 bits, con una banda totale di 528MB/s. A partire dal Pentium, il bus PCI è utilizzato insieme al bus ISA e al bus dedicato al collegamento con la memoria.

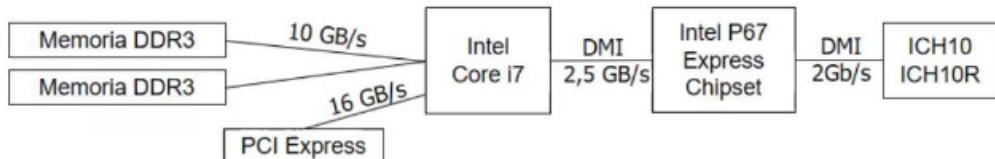


Alla fine degli anni 90 fu introdotto un bus dedicato per le schede grafiche l'AGP (Accelerated Graphics Port) che funzionava a 2,1 GB/sec. Nel P4 esiste un bridge che

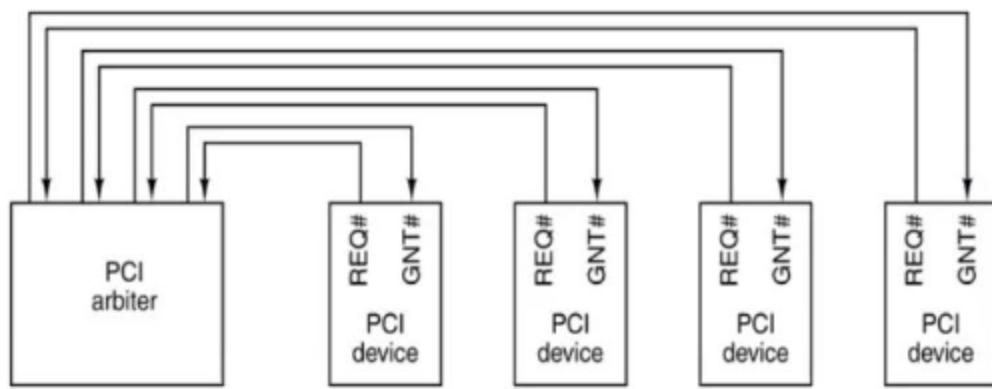
collega tutti i dispositivi ed è suddiviso in due sub-bridge collegati con una interconnessione veloce: Una connette CPU, memoria e controller video, L'altra il controller ATAPI (HD e DVD) e il bus PCI (SCSI e USB2).

PCIe

Il bus PCIe (PCI Express) può arrivare fino a 16 GB/s su collegamenti seriali ad alta velocità. In un sistema basato su Core i7 molte interfacce sono integrate sul chip della CPU: Due canali di memoria a 1,3 GHz hanno una larghezza di banda di 10GB/s. Un canale PCIe a 16 corsie con una larghezza di banda di 16 GB/s verso l'I/O. Il bridge (Intel P67) in questo caso è collegato alla CPU tramite un interfaccia seriale DMI ed in grado di collegare dispositivi di I/O veloci. Il chip (ICH10cattura) permette di collegare dispositivi di I/O più lenti o più vecchi.



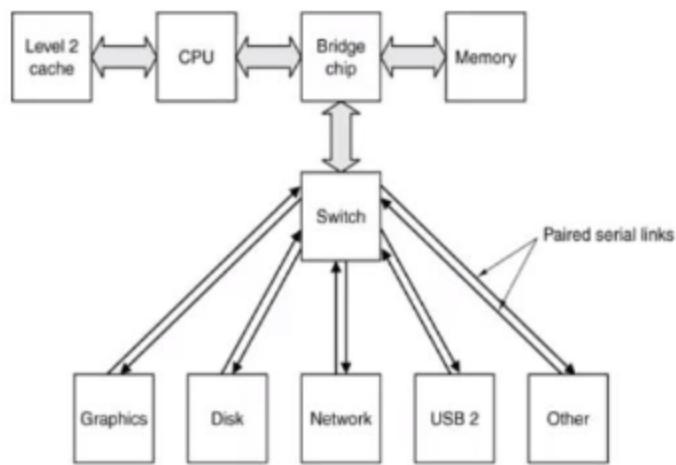
Il bus PCI utilizza l'arbitraggio centralizzato e l'arbitro è di solito inserito in uno dei chip di bridge, ogni dispositivo ha due linee che lo connettono all'arbitro: *REQ#* è utilizzato per richiedere il bus. *GNT#* è usato per la conferma della richiesta.



Il bus PCIe cambia il concetto di bus parallelo (usato da ISA/EISA/PCI) proponendo un'architettura basata su connessioni seriali punto-punto. La CPU, memoria e la cache sono connesse al chip di bridge nel modo tradizionale. Il cuore dell'architettura è uno

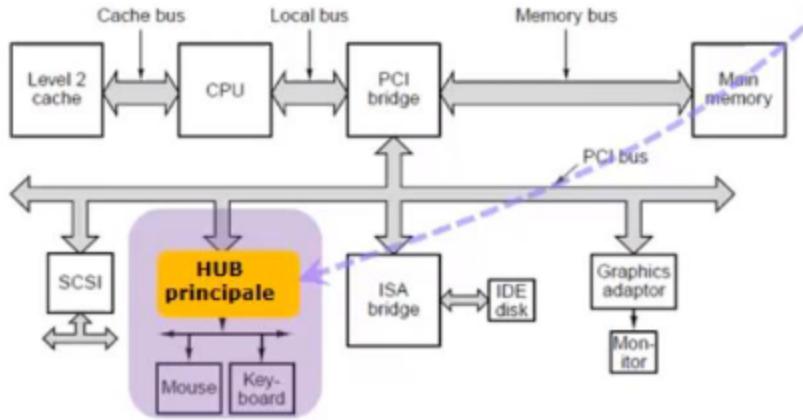
switch: una connessione dedicata punto-punto è realizzata per ogni chip di I/O. Ogni connessione si compone di una coppia di canali unidirezionali, ciascuno dei quali ha due cavi (segnale e massa). Il tipo di connessione è master-slave: il master invia un pacchetto dati allo slave (come accade nel mondo delle reti). Ogni pacchetto contiene informazioni di controllo (intestazione), eliminando così i segnali di controllo del bus PCI, e dati che devono essere trasferiti (payload). Un codice di correzione degli errori è aggiunto ai pacchetti.

In effetti in un PC con un bus PCIe abbiamo una mini rete a commutazione di pacchetto. La connessione tra switch e dispositivi non può eccedere i 50 cm. Il sistema è espandibile poiché si può collegare un altro switch al posto di un dispositivo creando così un albero di switch (max 3). I dispositivi sono inseriti o rimossi quando il sistema è in funzione, i connettori seriali più piccoli dei connettori paralleli, quindi risultano più piccolo anche i dispositivi e i computer.



USB

Il bus PCI e PCIe sono ottimi per connetter periferiche ad alta velocità, ma sono troppo costosi per quelle a bassa velocità. USB (Universal Serial Bus) è stato standardizzato nel 1998 per il collegamento con dispositivi lenti, la versione 1.0 ha una banda di 1,5MBps, la 2.0 480MBps, mentre la 3.1 arriva a 10GBps. Un sistema USB si compone di un hub principale (o root hub) connesso al bus di sistema dove, a sua volta, si possono collegare le periferiche o altri hub.



Il cavo di collegamenti di USB 1.1 : 2.0 si compone di 4 fili: due per i dati, uno di alimentazione e uno di massa. USB 3.0 ha invece 10 fili. Quando un nuovo dispositivo viene collegato, l'hub root rileva questo evento genera un'interruzione per il sistema operativo che:

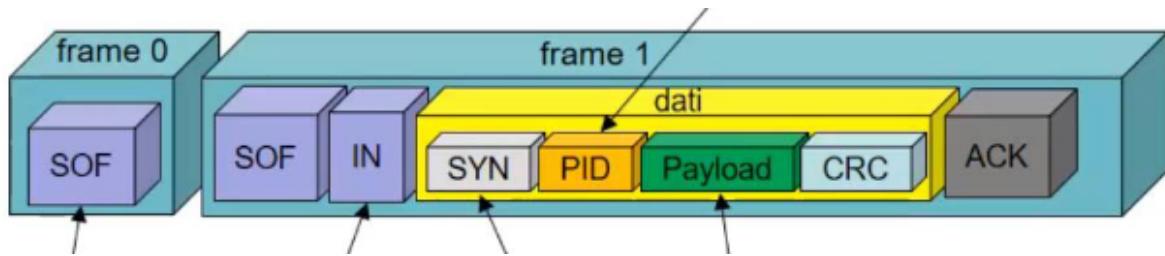
- Interroga il dispositivo per sapere di che tipo di periferica si tratta e di che banda ha bisogno.
- Se la larghezza di banda è sufficiente, il sistema operativo gli assegna un numero unico (1:127) e carica le informazioni del dispositivo.
- Ora è pronto per funzionare e non sono necessarie altre operazioni.

L'hub effettua il collegamento punto-punto con i dispositivi di I/O come se ci fossero dei tubi. Per mantenere sincronismo, l'hub ogni ms spedisce broadcast un nuovo frame (anche vuoto). L'USB supporta quattro tipi di frame:

- Controllo: utilizzati per configurare il dispositivo, inviargli comandi, interrogarlo sullo stato.
- Isocroni: utilizzati per dispositivi real-time come microfoni o telefoni che necessitano di spedire o accettare dati ad intervalli di tempo precisi.
- Bulk: utilizzati per grandi trasferimenti di dati come nel caso delle stampanti che non richiedono funzionamento in tempo-reale.
- Interrupt: sono fondamentali in quanto USB non supporta il concetto di interruzione quindi il sistema operativo senza di essi sarebbe costretto ad interrogare in polling il dispositivo.

Un frame contiene uno o più pacchetti:

- token: utilizzati il controllo del sistema dall'hub al device:
- dati: 8 bit di sincronizzazione, identificatore del tipo pacchetto (PID), payload e CRC (16 bit).
- handshake: ACK (l'hub ha ricevuto bene), NAK (c'è stato un errore di CRC) e STALL (attendere).
- speciali: utilizzati per usi specifici



3.g Interfacce

Interfacce di I/O

Le interfacce di I/O sono le schede che permettono ai dispositivi di I/O di collegarsi sul bus e di scambiare dati all'interno del computer. Esistono dieci chip standard per la realizzazione dei controllori:

- UART (Universal Asynchronous Receiver Transmitter), legge un byte dal bus e trasmette un bit alla volta su una linea seriale (utilizzata in passato per il collegamento con i terminali) oppure compie il lavoro opposto.
- USART (Synchronous UART), aggiungono alle UART la possibilità di effettuare trasmissioni sincrone.
- PIO (Parallel I/O), chip per il collegamento di un dispositivo di I/O con una comunicazione parallela.

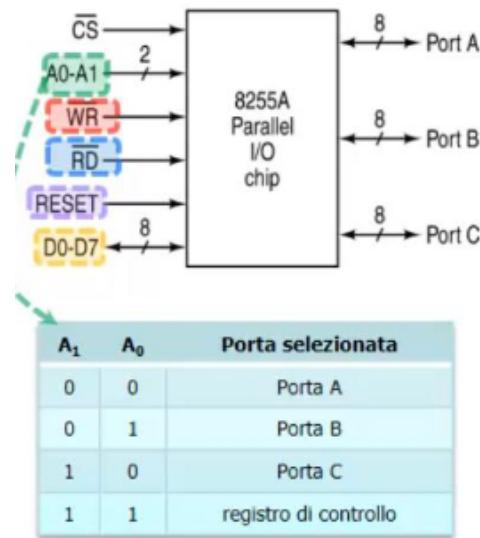
Un esempio di PIO: Intel 8255A ha 3 porte I/O (A, B, C) ciascuna da 8-bit con associato un registro latch più un registro interno. L'ingresso per l'abilitazione del chip è usato per collegare più PIO in parallelo. Due linee di indirizzamento della porta o del registro interno (A).

Il segnale RD indica che la CPU sta effettuando una lettura dal bus dati.

Il segnale WR indica che la CPU ha emesso i dati sul bus e sono validi per una operazione di scrittura.

Il segnale di RESET

8 linee 3-state per il collegamento al bus dati (D).



Decodifica dell'indirizzo

I dispositivi di I/O possono essere indirizzati in due modi:

1. **port-mapped I/O o I/O isolato** ovvero come un dispositivo di I/O reale.

è necessaria una linea del control bus che distingue se l'operazione deve essere eseguita in memoria oppure su I/O.

sono utilizzate delle istruzioni specifiche (es. IN e OUT).

2. **memory-mapped I/O** ovvero come parte della memoria.

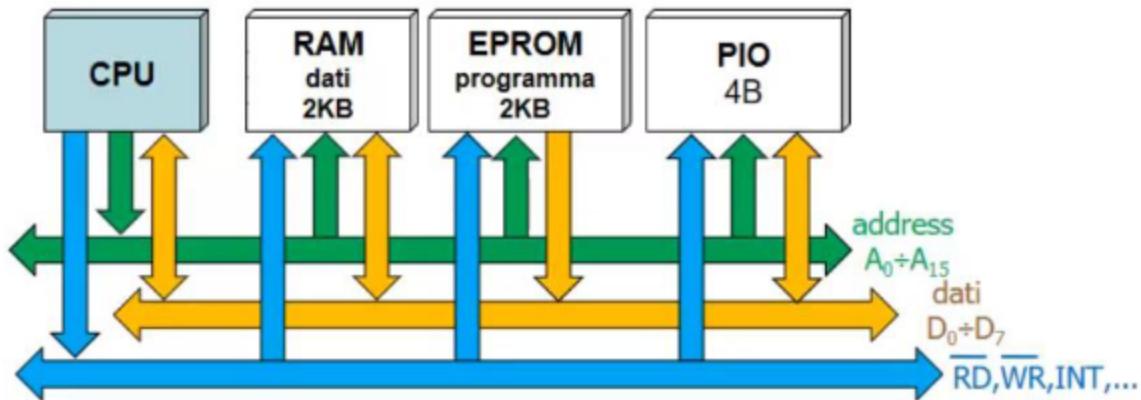
occorre riservare uno spazio in memoria che sarà destinato all'I/O.

le operazioni di lettura e scrittura in memoria eseguite in quello spazio di indirizzamento saranno dirottate sull'I/O.

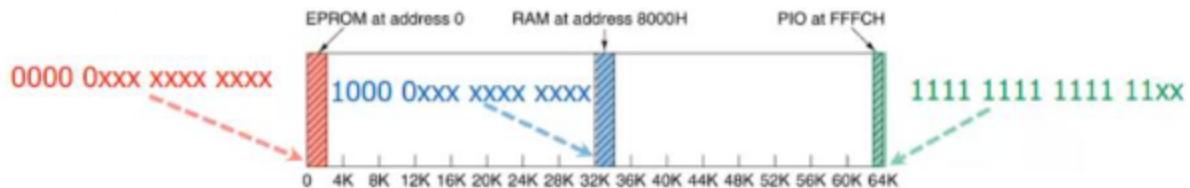
Si supponga di avere un calcolatore:

- Monoprocessoresso con 16-bit di indirizzamento (A) e un bus dati a 8-bit (D).
- Una EPROM di 2KB x 8 byte per il programma.
- Una RAM di 2KB x 8 byte per i dati.

- Una PIO tipo Intel 8255A con 3 porte e un registro di controllo.



Lo spazio di indirizzamento è $2^{16} = 64KB$. Si può scegliere di allocare il programma e i dati in un qualsiasi segmento da 2KB, mentre alla PIO sono necessari solo 4 byte.

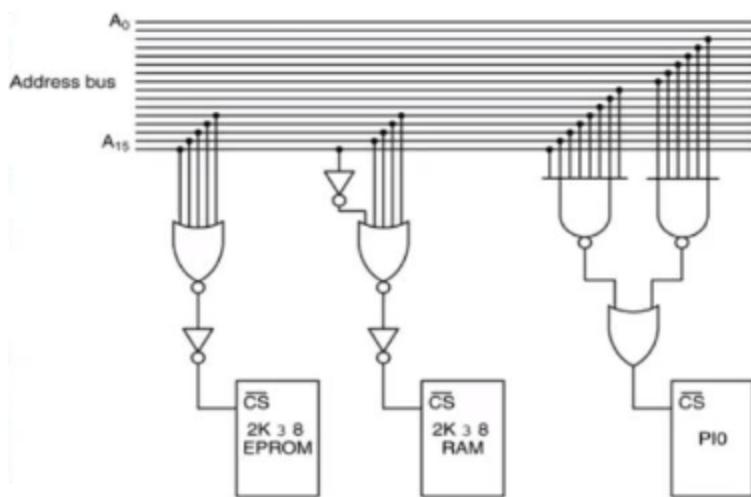


EPROM risponde quando $A_{15} : A_{11}$ sono bassi cioè nell'intervallo $0000_H : 07FF_H$

RAM risponde quando A_{15} è alto e $A_{14} : A_{11}$ sono bassi cioè $8000_H : 87FF_H$

PIO risponde quando $A_{15} : A_2$ sono alti cioè $FFFCH : FFFF_H$.

L'abilitazione del chip può essere realizzato con l'utilizzo delle porte logiche



4. Livello di microarchitettura

Il modo in cui viene progettato il livello di microarchitettura non dipende solamente dall'ISA che si intende implementare, ma anche dagli obiettivi di costi e prestazioni del calcolatore. Molti ISA moderni in particolare RISC, sono costituiti da istruzioni semplici che generalmente è possibile eseguire in un unico clock. Nel caso di ISA più complessi l'esecuzione di una singola istruzione può invece richiedere più cicli. Per eseguire un'istruzione è necessario localizzare gli operandi all'interno della memoria, leggerli e infine memorizzare il risultato nuovamente in memoria.

4.a Esempio di microarchitettura

Ogni ISA ha i propri principi generali. Un esempio di ISA è la JVM (Java Virtual Machine) che contiene istruzioni complesse implementate mediante la microprogrammazione IJVM lavora solo con interi. Questa microarchitettura conterrà un microprogramma (registrato in una ROM) il cui compito sarà prelevare, decodificare ed eseguire le istruzioni. Un modello convenzionale per progettare una microarchitettura consiste nel concepirla come un problema di programmazione, in cui ogni istruzione del livello ISA è una funzione che deve essere richiamata al programma principale. Il microprogramma si compone di una sequenza di microistruzioni che utilizzano variabili che costituiscono lo stato della macchina. Esso ha delle variabili che costituiscono lo stato del calcolatore. Il Program Counter (PC) contiene il riferimento della prossima microistruzione da eseguire. Ogni microistruzione si compone di:

- Codice operativo (Opcode) che identifica il tipo di istruzione.
- Operandi su cui si applicherà l'istruzione.

Il modello di esecuzione adottato è basato su tre fasi:

- fetch, caricamento in memoria dell'istruzione
- decode, riconoscimento del tipo di istruzione da eseguire
- esecuzione, svolgimento del compito

Percorso dati

Il data path è quella parte della CPU che contiene l'ALU, i registri interni, gli input e output. Un insieme di registri controlla l'accesso in memoria.

Si compone di due bus: B e C (in quanto A è quello contenuto nel registro H). Alla base dell'ALU troviamo uno shifter.

Esistono due segnali di controllo:

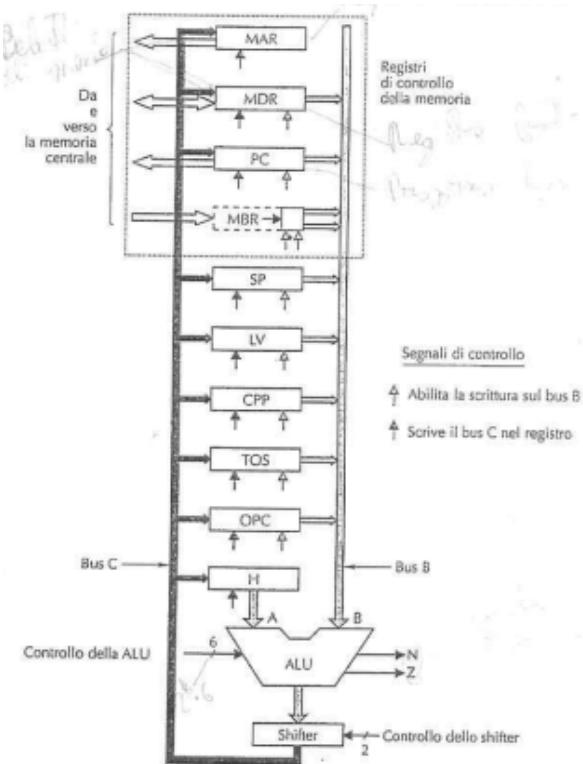
- abilita la scrittura del registro sul bus B.
- scrive il contenuto del bus C sul registro.

I registri sono:

- Memory Address Register (MAR).
- Memory Data Register (MDR).
- Program Counter (PC).
- Memory Byte Register (MBR), è un byte nello stream di istruzioni che provengono dalla memoria
- Stack Pointer (SP).
- Local Variable (LV), il riferimento nello stack alla base delle variabili locali.

Gli altri registri sono:

- Costant Pool (CPP).
- Top word On the Stack (TOS).
- Op Code register (OPC): registro temporaneo, può contenere l'ultima istruzione eseguita prima di un salto.
- Holding (H).



La ALU ha sei linee di controllo:

- F0 e F1 selezionano il tipo di funzione
- ENx, abilità il valore della variabile x altrimenti lo annulla
- INVA, esegue una sottrazione della variabile A
- INC, incrementa.

F ₀	F ₁	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	\bar{A}
1	0	1	1	0	0	\bar{B}
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B - A
1	1	0	1	1	0	B - 1
1	1	1	0	1	1	-A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	-1

La ALU agisce su due operandi: uno viene dal registro H e l'altro dal bus B. è possibile "spostare" l'operando da B ad A, applicando la funzione che restituisce l'operando B e successivamente memorizzando il risultato del bus C in H. Lo shifter può far scorrere i bit/byte (aritmetico/logico) del risultato verso destra o sinistra.

Temporizzazione del data path

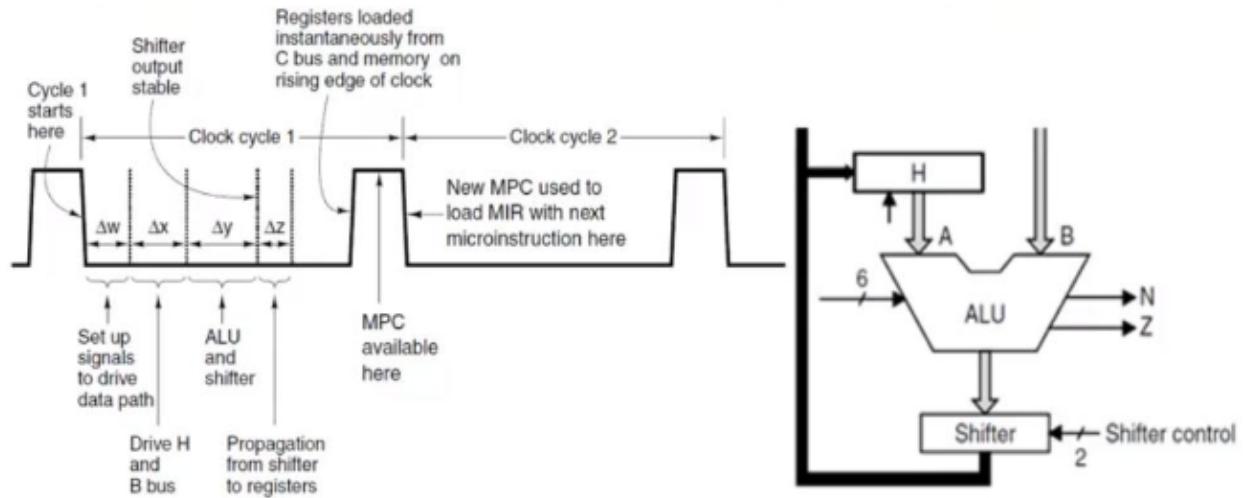
All'inizio di ogni ciclo di clock viene generato un breve impulso che può essere determinato dal clock principale. In corrispondenza del fronte in discesa dell'impulso vengono impostati i bit che piloteranno tutte le porte logiche. Questa Operazione richiede un intervallo di tempo W che deve essere conosciuto a priori.

Il registro richiesto viene selezionato e il suo contenuto viene portato sul bus B (prima che il suo valore diventi stabile bisogna aspettare un tempo X).

A questo punto la ALU e lo shifter hanno dati validi su cui operare. I loro output diventano stabili dopo un'ulteriore intervallo di tempo Y. Passato altro tempo Z, i risultati vengono propagati lungo il bus C fino ai registri in cui possono essere caricati in corrispondenza del fronte in salita dell'impulso successivo. Per l'incremento di un registro

- Si pone il valore del registro sul bus B.
- Si disabilita l'operando A e si incrementa B
- Non si effettua alcun scorrimento

- Si riscrive il risultato nel registro originario



Microistruzioni

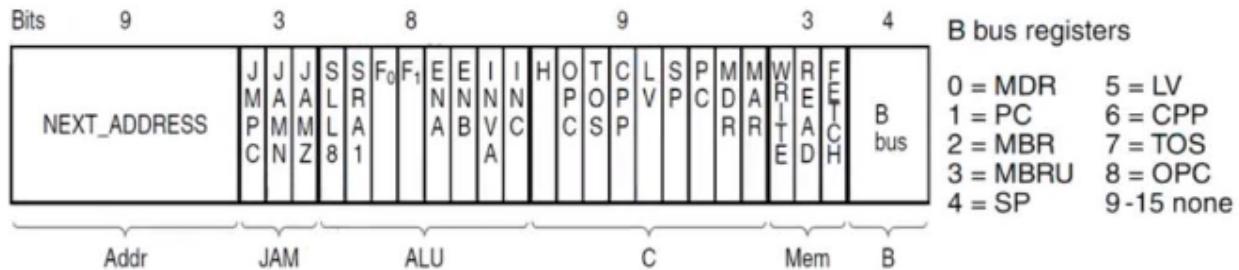
Per controllare il percorso dati abbiamo bisogno di 29 segnali suddivisibili in 5 gruppi funzionali:

- 9 segnali per controllare la scrittura dei dati dal bus C all'interno dei registri
- 9 segnali per controllare l'abilitazione dei registri del bus B per l'input all'ALU
- 8 segnali per controllare le funzioni della ALU e dello shifter
- 2 segnali (non mostrati) per indicare alla memoria di leggere o scrivere attraverso i registri MAR o MDR
- 1 segnale (non mostrato) per indicare il prelievo della memoria attraverso il PC o MBR

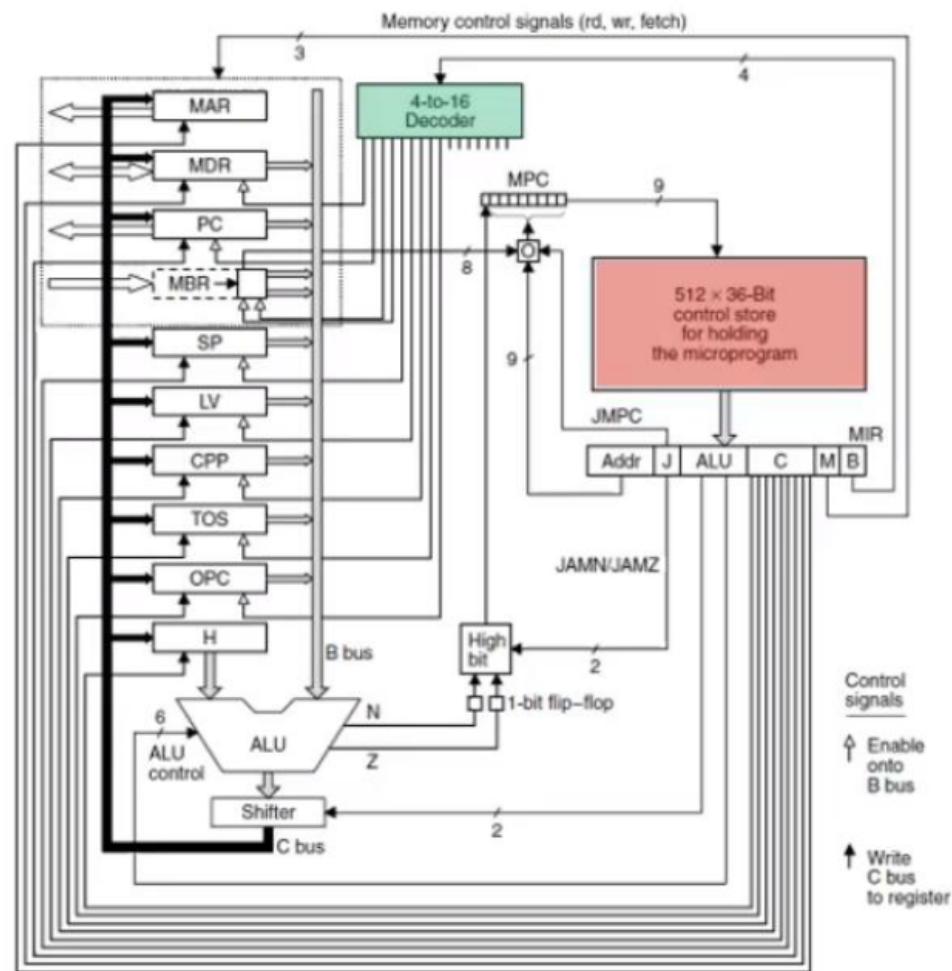
I valori di questi 29 segnali specificano le operazioni (cioè le microistruzioni) da eseguire durante un ciclo del percorso dati. Una microistruzione è un a sequenza di bit che costituisce un microprogramma cioè una sequenza di cicli nel datapath necessaria all'esecuzione dell'istruzione. Il formato delle istruzioni è rappresentato così:

- Addr, è l'indirizzo della potenziale successiva microistruzione
- JAM, determina come viene selezionata la prossima microistruzione
- ALU, seleziona le funzioni dell'ALU e dello shifter

- C, seleziona quali registri sono scritti dal bus C.
- Mem, seleziona la funzione in memoria
- B, seleziona quale registro è scritto sul bus B.



Mic-1



Il Mic-1 è una microarchitettura che contiene un sequenziatore che ha la responsabilità di far avanzare passo passo la sequenza di operazione necessaria per eseguire una singola istruzione. Durante ogni ciclo il sequenziatore deve produrre due tipi di informazione:

- lo stato di ogni segnale di controllo del sistema
- l'indirizzo della microistruzione da eseguire subito dopo.

La memoria di controllo memorizza le microistruzioni. Il decoder decodifica il tipo di istruzione in base al opcode. Le istruzioni in memoria centrale sono eseguite nell'ordine di memorizzazione (a meno di salti). Ciascuna microistruzione determina la successiva istruzione (non hanno necessariamente una memorizzazione lineare). La memoria di controllo ha un proprio registro indirizzo (MPC) e dati (MIR). I segnali si propagano nel data path ed un registro viene inserito nel bus B e la ALU sa quale operazione compiere. Una volta che gli input della ALU sono stabili, esegue il calcolo ed i segnali si propagano in uscita. Quando i segnali sono stabili, il bus C viene memorizzato all'interno dei registri candidati in corrispondenza del fronte in salita del clock (e termina così un ciclo).

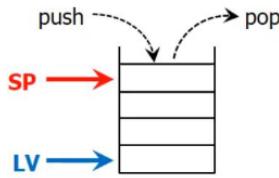
4.b Esempio di ISA: IJVM

Il livello ISA descrive l'architettura delle istruzioni che la CPU è in grado di eseguire in Hardware (Firmware). Ogni diversa CPU ha un proprio ISA e quindi istruzioni diverse spesso non compatibili tra loro.

Ogni istruzione di livello ISA è una funzione che deve essere richiamata dal programma principale. Il PC (Program Counter) è una delle variabili dello stato e indica la locazione di memoria che contiene la successiva istruzione ISA da eseguire.

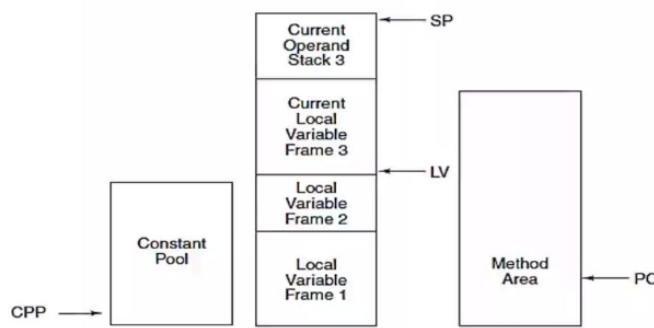
Stack

Lo stack è una struttura utilizzata per memorizzare lo stato di procedure che segue la filosofia LIFO (Last-In-First-Out). Questa organizzazione dei dati permette di gestire anche situazioni di chiamate ricorsive



Modello di memoria

La memoria può essere vista come un vettore: 4GB, l'unico modo che le istruzioni IJVM hanno per accedere alla memoria è quello di indicizzarla utilizzando puntatori.



L'unico modo che le istruzioni IJVM hanno per accedere alla memoria è quello di indicizzarla utilizzando dei puntatori. In ogni momento sono definite le seguenti aree di memoria:

- *Porzione costante di memoria.* I programmi IJVM non possono scrivere in quest'area che contiene costanti, stringhe e puntatori ad altre aree di memoria a cui è possibile far riferimento. E' caricata quando il programma è portato in memoria e in seguito non viene modificata.
- *Blocco delle variabili locali.* Per ogni invocazione di un metodo viene allocata un'area in cui memorizzare le variabili locali durante l'intero ciclo di vita dell'invocazione. Nella parte iniziale di questo blocco sono memorizzati i parametri (argomenti) con cui è stato invocato il metodo. Il blocco delle variabili locali non comprende lo stack degli operandi che è separato.
- *Stack degli operandi.* Il blocco dello stack non può superare una certa dimensione, stabilita in anticipo dal compilatore java. Lo spazio per lo stack degli operandi è allocato direttamente sopra il blocco delle variabili locali.

- *Area dei metodi.* Infine c'è una regione di memoria in cui risiede il programma. Qui è presente un registro隐式 che contiene l'indirizzo della successiva istruzione da prelevare. Questo puntatore è chiamato (Program Counter) PC.

Insieme d'istruzioni

Hex	Mnemonic	Meaning
0x10	BIPUSH <i>byte</i>	Push byte onto stack
0x59	DUP	Copy top word on stack and push onto stack
0xA7	GOTO <i>offset</i>	Unconditional branch
0x60	IADD	Pop two words from stack; push their sum
0x7E	IAND	Pop two words from stack; push Boolean AND
0x99	IFEQ <i>offset</i>	Pop word from stack and branch if it is zero
0x9B	IFLT <i>offset</i>	Pop word from stack and branch if it is less than zero
0x9F	IF_ICMPEQ <i>offset</i>	Pop two words from stack; branch if equal
0x84	IINC <i>varnum const</i>	Add a constant to a local variable
0x15	ILOAD <i>varnum</i>	Push local variable onto stack
0xB6	INVOKEVIRTUAL <i>disp</i>	Invoke a method
0x80	IOR	Pop two words from stack; push Boolean OR
0xAC	IRETURN	Return from method with integer value
0x36	ISTORE <i>varnum</i>	Pop word from stack and store in local variable
0x64	ISUB	Pop two words from stack; push their difference
0x13	LDC_W <i>index</i>	Push constant from constant pool onto stack
0x00	NOP	Do nothing
0x57	POP	Delete word on top of stack
0x5F	SWAP	Swap the two top words on the stack
0xC4	WIDE	Prefix instruction; next instruction has a 16-bit index

5. Il livello di macroarchitettura

Il livello ISA si trova tra quello della microarchitettura e il sistema operativo. Il livello ISA fu il primo a essere sviluppato, esso costituisce l'interfaccia tra il software e l'hardware. Un buon ISA dovrebbe definire un insieme di istruzioni che può essere implementato efficientemente da tecnologie presenti e future, dovrebbe inoltre favorire una compilazione del codice pulita, la regolarità e la completezza del ventaglio di scelte disponibili per il compilatore costituiscono un tratto importante, altrimenti il compilatore

potrebbe trovarsi in difficoltà con alcune operazioni importanti utilizzando operazioni limitanti.

5.a Overview del livello ISA

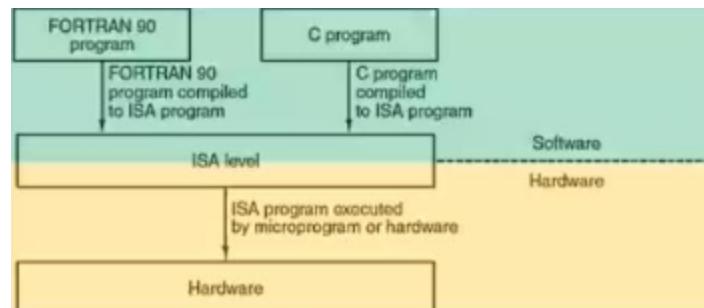
Proprietà del livello ISA

Il livello di architettura dell'insieme di istruzioni (ISA) è la macchina dal punto di vista del programmatore in linguaggio macchina. L'ISA si compone:

- Del modello di memoria.
- Dell'insieme dei registri.
- Dei tipi di dati possibili.
- Dell'insieme delle istruzioni.

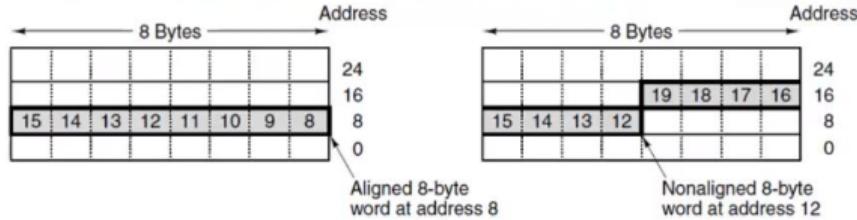
Dal livello ISA non è possibile conoscere il funzionamento della microarchitettura (parallelismo, pipeline, ...). Normalmente il livello ISA è descritto attraverso un documento formale di definizione del produttore. È l'interfaccia tra i compilatori e l'hardware quindi il linguaggio che entrambi possono comprendere. Deve essere retrocompatibile: cioè essere in grado di far girare vecchi programmi. Normalmente ha due modalità operative:

- Modalità Kernel - Per eseguire il SO e tutte le istruzioni.
- Modalità Utente - Per eseguire i programmi utente e non operazioni "sensibili" (come quelle che accadono alla cache).



Modelli di memoria

Tutti i computer suddividono la memoria in celle adiacenti di byte che sono a loro volta raggruppati in gruppi di 4 (32 bit) o 8 (64 bit). Le parole possono essere allineate all'indirizzo base



I processori a livello ISA normalmente dispongono di uno spazio di memoria lineare (2^{32} o 2^{64}), talvolta alcuni hanno una suddivisione tra dati e istruzioni (questo rende più difficili gli attacchi di malware).

Registri

Tutti i computer hanno dei registri visibili a livello ISA, alcuni registri del livello sottostante non sono visibili come TOS e MAR. Il loro compito è il controllo dell'esecuzione del programma, il contenimento dei risultati temporanei o altro. I registri ISA si possono dividere in due categorie:

- Specializzati: Program Counter, Stack Pointer e quelli visibili solo in modalità kernel (controllo della cache, la memoria, i dispositivi di IO e altre funzionalità hardware).
- Di uso generale: sono utilizzati per memorizzare i risultati temporanei delle variabili locali.

Il registro dei Flag (Program Status Word - PSW) è un registro ibrido poiché è tra la modalità kernel e la modalità utente. Questo registro di controllo contiene vari bit che sono necessari alla CPU e che vengono impostati a ogni ciclo dell'ALU e riflettono lo stato del risultato dell'operazione più recente.

Questi sono:

- N: posto a 1 dopo risultato è negativo.
- Z: posto a 1 dopo risultato uguale a 0.
- V: posto a 1 se il risultato ha causato un overflow.
- C: posto a 1 se il risultato ha causato un riporto oltre l'ultimo bit più significativo.

- A: posto a 1 se si è verificato un riporto oltre il terzo bit (riporto ausiliario).
- P: posto a 1 se il risultato è pari.

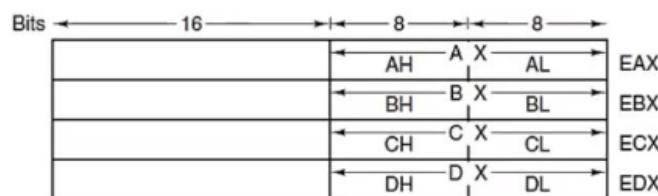
Panoramica del livello ISA del Core i7

Mantiene la compatibilità fino all'8086 e 8088 (anni '70) a loro volta basati su 4004. Fino all'80286 il bus indirizzi era a 16 bit e si potevano indirizzare 16.384 segmenti da 64KB. Dal x386 nasce una nuova architettura denominata IA-32 su cui si basano tutti gli attuali processori Intel. I cambiamenti introdotti dal x386 sono le istruzioni MMX, SSE e SSE2 nate per applicazioni multimediali. Un'altra evoluzione importante è l'ampliamento del bus a 64 bit (x86-64). Il Core i7 ha 3 diverse modalità operative:

- Modalità reale: si comporta come un 8088 ma nel caso vengono eseguite istruzioni errate la macchina va in blocco.
- Modalità virtuale: permette di eseguire programmi 8088 in modo protetto e controllato da un vero SO.
- Modalità protetta: si comporta come un Core i7 con 4 privilegi controllati da PSW:
 - Livello 0 (o modalità kernel) - Usato dal SO.
 - Livello 1 e 2 - Usati raramente.
 - Livello 3 - Usato dai programmi utente in modo protetto.

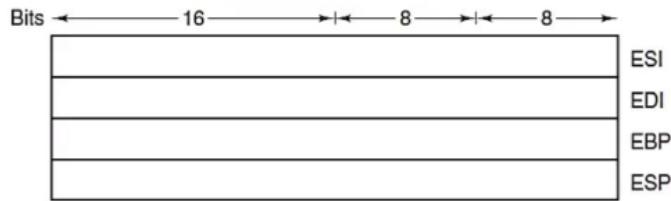
La memoria è divisa in 16384 segmenti da 64KB anche se la maggior parte dei SO supporta un solo segmento. Il core i7 ha 4 registri di uso generale (tutti a 32 bit):

- EAX: usato per le operazioni aritmetiche.
- EBX: usato come puntatore a indirizzi di memoria.
- ECX: usato come contatore nei cicli.
- EDX: usato nelle moltiplicazioni/divisioni durante le quali contiene con EAX prodotti/dividendi a 64 bit.



Tutti questi registri possono essere utilizzati a 16 o 8 bit. Ulteriori 4 registri a 32 bit con caratteristiche specifiche sono:

- ESI: puntatore in memoria alla stringa sorgente.
- EDI: puntatore in memoria alla stringa destinazione.
- EBP: riferenzia l'indirizzo base del record di attivazione corrente (analogo al registro LV della IJVM).
- ESP: Puntatore allo stack (come SP di IJVM).



Altri registri sono:

- I registri segmento (tutti a 16 bit) che derivano dalla compatibilità dell'indirizzamento a 16 bit dell'8088: CS, SS, DS, ES, FS e GS.
- Il Program Counter che è EIP (Extender Instruction Pointer).
- L'insieme dei bit dei flag della program status word EFLAGS.

5.b Tipi di dati

Tutti i computer hanno bisogno di dati. All'interno del computer, i dati devono essere rappresentati in una forma specifica. A livello ISA sono disponibili una varietà di tipi di dati diversi che esamineremo in seguito.

Tipi di dati numerici

I tipi di dati possono numerici è costruito dagli interi. Ci sono interi di lunghezze diverse, in genere 8, 16, 32, 64 bit. Gli interi servono a contare oggetti, per identificare oggetti e vanno da $-2^{n-1} : 2^{n-1}$ con segno dove il bit in posizione meno significativa è 0 se positivo o 1 se negativo e 0 : 2^n senza segno. Esistono anche i numeri reali rappresentati tramite numeri in virgola mobile lunghi da 32, 64, 128 bit.

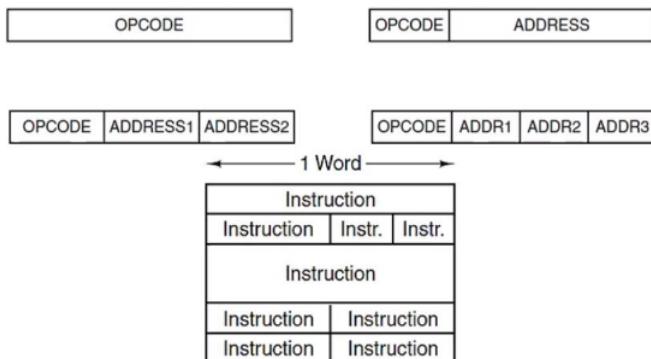
Tipi di dati non numerici

Caratteri (ASCII e UNICODE) ovvero sequenze di stringhe o caratteri, BitMap, Puntatori (SO, PC, LV, CPP) cioè un indirizzo di macchina. Anche numeri Booleani, 0 o 1 True o False.

5.c Formati di istruzioni

Una istruzione si compone di:

- codice operativo dell'istruzione (Opcode).
- indirizzo di riferimento degli operandi (opzionali).



L'argomento generale che tratta dalla provenienza degli operandi è detto indirizzamento.

Criteri progettuali per i formati d'istruzioni

Istruzioni corte sono preferibili alle lunghe per vari ordini di ragioni:

- Permette di avere programma più piccoli a parità di istruzioni.
- Permette di memorizzare maggiori quantità di istruzioni nelle cache dei processori che, notoriamente, hanno una larghezza di banda (bit/sec) limitata rispetto alla capacità di calcolo dei processori.

Minimizzare troppo la dimensione può causare la difficoltà nella decodifica. Il criterio di ottenere istruzioni di dimensione minima deve essere valutato in base al tempo richiesto per la decodifica e l'esecuzione delle istruzioni. Un'altra ragione per minimizzare la

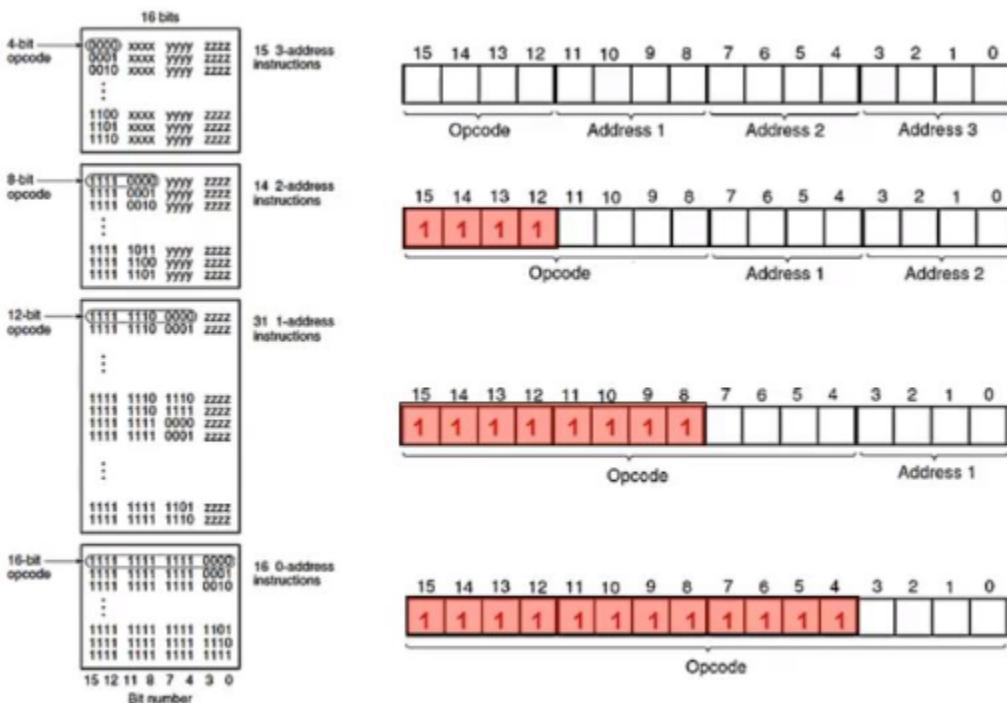
dimensione delle istruzioni è che assumerà maggior rilievo al crescere delle prestazioni dei processori.

Un altro criterio progettuale prevede nel formato delle istruzioni spazio sufficiente a esprimere tutte le operazioni volute.

Un terzo criterio concerne il numero di bit in un campo degli indirizzi, per l'indirizzamento della memoria, uno spazio di indirizzamento ampio conduce ad istruzioni lunghe.

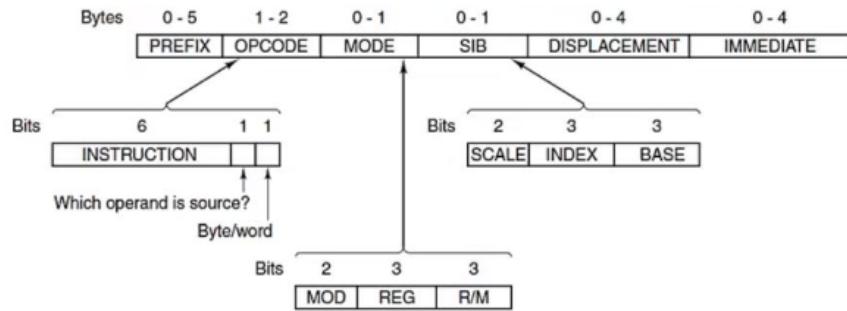
Codice operativo espandibile

L'idea di codice operativo espandibile evidenzia il compromesso che sussiste tra lo spazio riservato all'opcode e lo spazio per le altre informazioni. La capacità di usare dimensioni di opcode variabili può venire sfruttata in due modi diversi: in primo luogo si può mantenere costante la dimensione delle istruzioni, assegnando gli opcode più corti alle istruzioni che hanno bisogno di più bit per specificare informazioni di altra natura; in secondo luogo, è possibile minimizzare la lunghezza media delle istruzioni, scegliendo gli opcode più corti possibile per le istruzioni comuni e lasciando i più lunghi per le istruzioni d'uso più raro.



Formati delle istruzioni del Core i7

Esistono 6 campi di cui 5 opzionali di lunghezza variabili, uno dei due operandi è sempre un registro, l'altro un registro o in memoria, MODE stabilisce la modalità di indirizzamento, l'indirizzo di memoria è l'offset di un segmento. SIB (Scale Index Base) è un bit supplementare.



5.d Indirizzamento

Modalità di indirizzamento

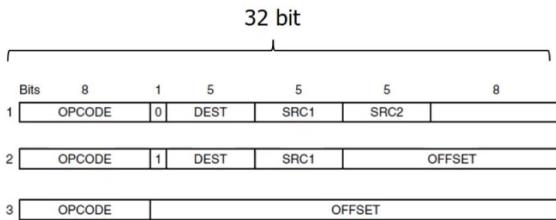
Esistono diverse modalità di indirizzamento:

- Immediato: il valore dell'operando è nell'istruzione.
- Diretto: l'istruzione contiene l'indirizzo di memoria completo dell'operando.
- Indiretto: l'indirizzo di memoria fornito contiene l'indirizzo dell'operando.
- A registro: si specifica un registro che contiene l'operando.
- Indiretto a registro: il registro specificato contiene l'indirizzo dell'operando.
- Indicizzato: l'indirizzo è dato da una costante più il contenuto di un registro.
- A registro base: viene sommato a tutti gli indirizzi il contenuto di registro.
- A stack: l'operando è sulla cima dello stack (o ci deve andare).

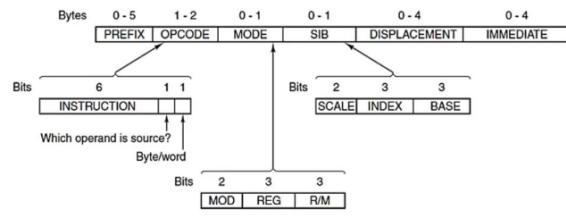
Ortogonalità

L'ortogonalità è il principio secondo cui ogni istruzione dovrebbe essere in grado di utilizzare qualsiasi modalità di indirizzamento supportata. L'ISA consente al programmatore di scegliere quella che soddisfa esattamente la necessità del proprio programma a quel punto, e quindi ridurre la necessità di utilizzare più istruzioni per raggiungere lo stesso scopo. Ciò significa che il numero totale di istruzioni viene ridotto,

risparmiando memoria e migliorando le prestazioni. Le architetture Intel non hanno queste caratteristiche.



Istruzione Ortogonale



Istruzione non ortogonale

5.e Tipi di istruzioni

Le istruzioni del livello ISA possono essere suddivise in gruppi facilmente rintracciabili.

Istruzioni di trasferimento dati

Poter copiare i dati da una locazione all'altra è fondamentale per tutte le operazioni. Per copia intendiamo la creazione di un nuovo oggetto costituito da una sequenza di bit identica all'originale. Le istruzioni di trasferimento dati dovrebbero chiamarsi “duplicazione di dati”. Ci sono due ragioni per copiare i dati da una locazione all'altra:
 L'assegnamento $A = B$;

Preparare i dati per un accesso e un uso differenti.

Queste istruzioni devono specificare in qualche maniera la quantità di dati da trasferire. In alcuni ISA esistono istruzioni per trasferire quantità di dati che partono da byte fino all'intera memoria. Altre macchina specificano solo gli indirizzi destinazione e sorgente senza indicare la quantità da trasferire.

Operazioni binarie

Le operazioni binarie sono quelle che producono un risultato dalla combinazione di due operandi. Tutti gli ISA hanno istruzioni per l'addizione e la sottrazione di interi. Un altro insieme di operazioni binarie comprende le istruzioni booleane. In genere sono disponibili AND OR NOT XOR NOR e NAND.

Operazioni unarie

Le operazioni unarie prendono in ingresso un operando e restituiscono un risultato. Sono più corte di quelle binarie. Le istruzioni per lo scorrimento o la rotazione del contenuto di una parola si rilevano molto utili. Ne fanno parte anche l'operazioni CLR (clear) che azzera il valore, NEG che lo nega (NOT), INC che lo incrementa di 1

Confronti e salti condizionati

Quasi tutti i programmi hanno bisogno della capacità di esaminare il contenuto dei dati e alterare la sequenza di esecuzione delle istruzioni in base al risultato dell'ispezione. Un metodo per farlo è fornire istruzioni di salto condizionato che verificano una certa condizione e saltano a un particolare indirizzo di memoria se la condizione è soddisfatta. Alle volte c'è un bit nell'istruzione che indica se il salto deve avvenire a condizione soddisfatta o non soddisfatta. Molte macchine hanno bit usati per specificare certe condizioni

Invocazione di procedure

Una procedura è un insieme d'istruzioni che svolge un certo compito e che può essere invocata da diversi punti in un programma. Il termine subroutine è usato spesso invece di procedure, anche chiamati funzioni o metodi. Quando una procedura ha terminato il proprio compito l'esecuzione deve riprendere dall'istruzione successiva alla chiamata. A tal fine l'indirizzo di ritorno deve essere passato alla procedura o salvato da qualche parte dove possa essere recuperato per fare ritorno. L'indirizzo di ritorno può essere salvato: in memoria, in un registro o nello stack. La procedura potrebbe chiamare altre procedure, la capacità di una procedura di richiamare se stessa è detta ricorsione.

Istruzioni di ciclo

Poiché capita spesso di dover eseguire un gruppo d'istruzioni un numero prefissato di volte, molte macchine dispongono di istruzioni per facilitare questo compito. Tutti gli schemi prevedono un contatore che viene incrementato o decrementato di un certo valore costante a ogni iterazione del ciclo. Il contatore viene anche esaminato a ogni iterazione, il ciclo termina quando si verifica una certa condizione.

Input/Output

Nessun raggruppamento d'istruzioni manifesta la stessa variabilità tra macchine diverse come le istruzioni di IO. I personal computer usano tre schemi diversi.

1. IO programmato con attesa attiva:

impiegato nei microprocessori di fascia bassa, una sola istruzione di input e una sola di output che trasferisce un carattere per volta da un registro al dispositivo.

Vengono utilizzati 4 registri 2 per lo stato e i dati in input e 2 per lo stato e i dati in output. Ha come principale svantaggio che la CPU passa gran parte del suo tempo in un ciclo serrato in cui attende che il dispositivo risulti pronto (attesa attiva).

2. IO interrupt driven:

per evitare l'attesa attiva la CPU fa partire il dispositivo di IO e gli impedisce l'ordine di generare un interrupt quando ha finito così da far lavorare costantemente la CPU e non bloccarla. Il problema è che ci vuole un interrupt per ogni carattere trasferito e l'elaborazione degli interrupt è gravosa.

3. IO con DMA:

La DMA è un nuovo chip con accesso diretto al bus che si occuperà di gestire il dispositivo di IO. In presenza di DMA la CPU deve solo inizializzare pochi registri, dopo di che è libera di svolgere il proprio lavoro fino al completamento del trasferimento segnalato da un interrupt proveniente da DMA. Succede però che il controllore DMA sottrae cicli di bus alla CPU, questo fenomeno viene detto cycle stealing.

Istruzioni del Core i7

L'insieme di istruzioni del Core i7 è una miscellanea di vere istruzioni di 32 bit e resti dell'8088. Molte istruzioni referenziano uno o due operandi, che possono trovarsi nei registri o in memoria. Il Core i6 dispone di parecchie istruzioni per caricamento, memorizzazione, trasferimento, confronto e scansione di stringhe di caratteri o di parole. Le istruzioni di questo Core prevedono molti prefissi.

5.f Controllo del flusso

Il controllo di flusso riguarda la sequenza con cui le istruzioni vengono eseguite dinamicamente, ovvero durante l'esecuzione del programma. In mancanza d'istruzioni di salto o di chiamate di procedura, le istruzioni vengono eseguite di norma nello stesso ordine con cui si susseguono in memoria. Le chiamate di procedura alterano il controllo del flusso, perché arrestano la procedura correntemente in esecuzione e cominciano l'esecuzione della procedura chiamata. Le coroutines sono legate alle procedure e

causano un'alterazione simile (sono utili nella simulazione di processi paralleli). Anche le trap e gli interrupt provocano un'alterazione del flusso esecutivo quando si verificano certe condizioni speciali.

Flusso sequenziale e diramazioni

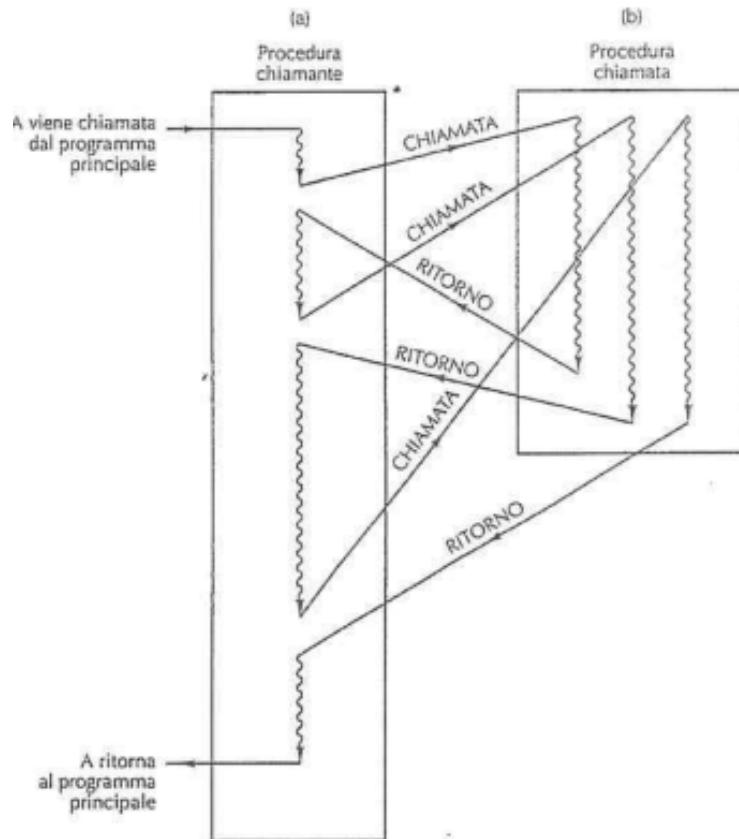
Molte istruzioni non alterano il controllo del flusso; dopo l'esecuzione di un'istruzione, viene recuperata ed eseguita l'istruzione che la segue in memoria. Al termine di ciascuna istruzione, il program counter viene incrementato della lunghezza dell'istruzione elaborata

Procedure

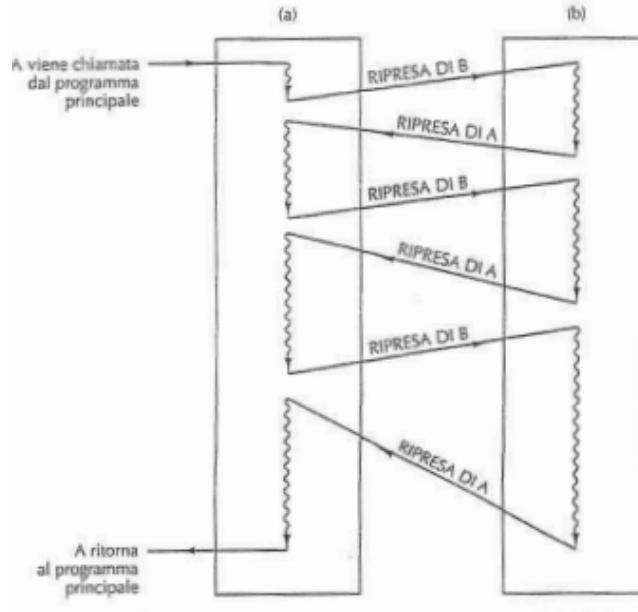
Le procedure costituiscono la tecnica più importante per la strutturazione dei programmi. Da un certo punto di vista una chiamata di procedura altera il flusso esecutivo tanto quanto un salto, ma a differenza di un salto alla terminazione del suo compito la procedura ripassa il controllo al comando o all'istruzione che segue la sua chiamata. Tuttavia, da un altro punto di vista, il corpo di una procedura può essere visto come la definizione di una nuova istruzione di alto livello. Così facendo, la chiamata di procedura è concepibile come un'istruzione singola, senza riguardo alla sua effettiva complessità. Ogni volta che una procedura viene chiamata, in cima allo stack viene allocato un record d'attivazione per la procedura stessa. Il record d'attivazione di creazione più recente è quello in uso corrente. Oltre al puntatore allo stack, che punta alla cima della pila, risulta spesso conveniente avere un puntatore al record d'attivazione, FP (frame pointer), che punta a una locazione nota del record d'attivazione. Le prime cose che una procedura invocata deve fare sono il salvataggio del vecchio FP (così che possa essere ripristinato all'uscita della procedura), la copia di SP in FP e l'eventuale incremento di FP di una parola, a seconda della locazione cui puntare nel nuovo record d'attivazione. Il punto chiave è poter effettuare un ritorno da procedura e ripristinare lo stato dello stack alla situazione in cui si trovava appena prima dell'invocazione della procedura corrente. Il codice che si preoccupa di salvare il vecchio FP, che stabilisce il nuovo FP e accresce il puntatore allo stack per riservare spazio alle variabili locali si chiama prologo della procedura (*procedure prolog*). All'uscita dalla procedura si rende invece necessario ripulire lo stack, che rappresenta la fase di epilogo della procedura.

Coroutine

Nell' usuale sequenza di chiamata c'è una chiara distinzione tra procedura chiamante e procedura chiamata. Ogniqualvolta A chiama B quest'ultima ricomincia dall'inizio, mentre A non ricomincia mai da capo, ma continua semplicemente ad andare avanti.



Questa differenza si riflette nel modo in cui il controllo è passato tra A e B. Per chiamare B, la procedura A si avvale dell'istruzione di chiamata di procedura, che memorizza l'indirizzo di ritorno (cioè l'indirizzo dell'istruzione che segue la chiamata) in un qualche luogo adatto allo scopo, per esempio in cima allo stack. Quindi pone l'indirizzo di B nel program counter, completando così la chiamata. Quando B ritorna, non usa l' istruzione di chiamata ma si avvale dell'istruzione di ritorno, che semplicemente fa il pop dell'indirizzo di ritorno dallo stack e lo copia nel program counter. Quando è invece A a passare il controllo a B, non salta all' inizio di B (tranne la prima volta) bensì all'istruzione successiva al "ritorno" più recente, vale a dire successiva alla chiamata di A più recente. Due procedure che si comportano in questo modo prendono il nome di coroutine.



Le coroutine sono usate comunemente per la simulazione dell'elaborazione parallela su singola CPU. Ogni coroutine gira in maniera pseudo-parallela alle altre, come se disponesse di una propria CPU

Trap

Una trap (“trappola”) è una specie di chiamata di procedura automatica effettuata quando si verificano certe condizioni causate da un programma e che sono in genere eventi rilevanti, ma di rara occorrenza. Ne è un buon esempio l’overflow. Il concetto chiave delle trap è che sono fatte scattare da condizioni eccezionali causate dal programma stesso e rilevate dall’hardware o dal microprogramma. Le trap fanno risparmiare tempo di esecuzione e memoria rispetto alla verifica affidata al controllo esplicito del programmatore. Alcune delle condizioni che causano comunemente trap sono gli overflow e gli underflow (interi o in virgola mobile), le violazioni di protezione, gli opcode non definiti, gli overflow di stack, i tentativi di utilizzare dispositivi di I/O inesistenti, i tentativi di fetch di una parola da un indirizzo dispari, la divisione per zero.

Interrupt

Gli interrupt (“interruzioni”) sono cambiamenti nel flusso esecutivo causati non dal programma in esecuzione, ma da qualche altro problema, in genere determinato dall’I/O. Come le trap, gli interrupt interrompono il programma in esecuzione e

trasferiscono il controllo a un gestore deputato a svolgere le azioni appropriate. Al loro compimento, il gestore di interrupt restituisce il controllo al programma interrotto. È suo compito far riprendere il processo interrotto esattamente dallo stesso stato in cui si trovava al momento dell'interruzione, il che implica il ripristino di tutti i registri interni allo stato precedente all'interrupt. La differenza essenziale tra le trap e gli interrupt è che le trap sono sincrone al programma e gli interrupt sono asincroni. Se un programma viene rieseguito un milione di volte con lo stesso input, le trap si verificheranno sempre nello stesso punto, mentre gli interrupt possono variare a seconda, per esempio, del momento in cui viene premuto il tasto d'invio al terminale. Un concetto chiave legato agli interrupt è la trasparenza. Al verificarsi di un interrupt si intraprendono alcune azioni e si procede con l'esecuzione di un certo codice, ma quando tutto è finito il computer deve ripartire esattamente dallo stato che aveva prima dell'interrupt. Una routine di interrupt che manifesta questa proprietà si dice trasparente.

5.g Architettura Intel IA-32 e IA-64

diventava sempre più difficile escogitare nuovi stratagemmi per velocizzare ulteriormente le implementazioni, dato che i vincoli imposti dall'ISA IA-32 si rivelavano sempre più costrittivi. Secondo alcuni ingegneri l'unica soluzione realistica era abbandonare IA-32 come linea principale di sviluppo e spostarsi verso un ISA completamente nuovo. È proprio ciò che Intel ha iniziato a fare: vi erano infatti progetti per due nuove linee. EMT-64 è un rifacimento ampliato del tradizionale ISA Pentium, con registri di 64 bit e spazio degli indirizzi a 64 bit. Questo nuovo ISA risolve il problema dello spazio degli indirizzi, ma presenta ancora le complicazioni implementative dei suoi predecessori. L'altra nuova architettura, sviluppata congiuntamente da Intel e Hewlett Packard, venne chiamata IA-64. Si tratta di una macchina completamente a 64 bit, non di un'estensione di una macchina a 32 bit. Inoltre si differenzia radicalmente dell'architettura IA-32 per molti aspetti. Inizialmente si rivolge al mercato dei server di fascia alta, ma Intel sperava di poter raggiungere anche il mercato dei desktop. Ciò non avvenne nonostante i suoi difetti, gli acquirenti rifiutarono di abbandonare l'architettura IA-32.

Il problema dell'ISA IA-32

IA-32 è un ISA datato, le cui proprietà sono inadatte alla tecnologia corrente: È un ISA CISC con istruzioni di lunghezza variabile e una miriade di formati differenti difficili da decodificare velocemente. La tecnologia attuale lavora meglio con ISA RISC

che hanno una sola lunghezza per le istruzioni e opcode di lunghezza fissa facili da decodificare.

IA-32 è un ISA orientato alla memoria ed è a due indirizzi. La maggior parte delle istruzioni referenzia la memoria e la maggior parte dei programmatore e dei compilatori non si cura di far riferimento continuo alla memoria.

IA-32 dispone di un insieme di registri piccolo e irregolare.

Per svolgere velocemente tutte queste operazioni occorre una pipeline con molti stadi. A sua volta, l'utilizzo di queste pipeline comporta che le istruzioni impieghino molti cicli per venir completate. Di conseguenza per garantire che le istruzioni introdotte nelle pipeline siano quelle giuste è essenziale una predizione dei salti molto precisa. Una predizione sbagliata richiede lo svuotamento della pipeline, un'operazione molto costosa, perciò anche un tasso di predizioni erronee abbastanza basso può causare un degrado sostanziale delle prestazioni.

Per alleviare il problema delle predizioni sbagliate il processore deve effettuare l'esecuzione speculativa, con tutti i problemi che comporta, specie quando i riferimenti alla memoria causano un'eccezione.

Modello IA-64 e calcolo parallelismo

L'idea chiave alla base di IA-64 è di spostare il carico di lavoro dalla fase di esecuzione alla fase di compilazione. Nel modello IA-64 il compilatore produce un programma che può essere eseguito così com'è. In questo modello il compilatore tiene traccia delle unità funzionali occupate e non emette istruzioni che usano unità funzionali non disponibili. Il modello che rende il parallelismo sottostante nell'hardware visibile al compilatore si chiama EPIC (*Explicitly Parallel Instruction Computing*). EPIC può essere considerato in un certo senso il successore di RISC. Il modello IA-64 presenta numerose funzionalità per incrementare le prestazioni. Tra queste ce ne sono alcune per ridurre i riferimenti in memoria, per lo scheduling delle istruzioni, per ridurre i salti condizionati e altro.

Riduzione degli accessi in memoria

Nei computer moderni l'accesso alla memoria costituisce un collo di bottiglia molto stretto, perché le CPU sono molto più veloci della memoria. Un modo per ridurre gli accessi alla memoria è disporre di una grande cache di primo livello sul chip e di una cache di secondo livello ancora più grande, vicina al chip. Oltre all'uso delle cache ci

sono altri modi per ridurre gli accessi in memoria, alcuni dei quali sono sfruttati da IA-64.

Scheduling delle istruzioni

Uno dei problemi principali del Core i7 è la difficoltà insita nel trovare uno scheduling delle varie istruzioni, a favore delle diverse unità funzionali, che eviti le dipendenze . Ci vogliono meccanismi esageratamente complicati per gestire tutti questi aspetti in fase d'esecuzione, e così un'ingente porzione dell'area del chip è dedicata al loro trattamento. L'idea chiave è che un programma consista in una sequenza di gruppi d'istruzioni. Entro un certo limite le istruzioni di un gruppo non sono mai in conflitto tra di loro, non usano più unità funzionali e risorse di quelle a disposizione della macchina. Una conseguenza di queste regole è che la CPU è libera di scegliere l'ordine di esecuzione delle istruzioni (inclusa l'esecuzione parallela), senza timore di conflitti. Le istruzioni sono organizzate in pacchetti d'istruzioni (bundle) di 128 bit.

Riduzione dei salti condizionati

Un'altra caratteristica importante di IA-64 è il modo nuovo di trattare i salti condizionati. Se ci fosse un modo di liberarsene quasi del tutto, le CPU ne beneficierebbero grandemente in semplicità e velocità. IA-64 usa una tecnica, detta attribuzione di predici (predication), che può ridurre di molto il loro numero. Le istruzioni contengono condizioni (predicati) che stabiliscono se devono essere eseguite o meno. Questo cambio di paradigma da istruzioni incondizionate a istruzioni predervative rende possibile la riduzione di (molti) salti condizionati. Tutte le istruzioni di IA-64 sono predervative. Ciò significa che l'esecuzione di ogni istruzione può essere resa condizionata.

Caricamenti speculativi

Un'altra caratteristica di IA-64 che ne velocizza l'esecuzione è la presenza di LOAD speculative. Se un'istruzione LOAD è speculativa e non va a buon fine, invece di causare un'eccezione, semplicemente si ferma e viene asserito un bit associato al registro da caricare che lo segnala come invalido. In più IA-64 è provvista di un modello di parallelismo esplicito che richiede un compilatore in grado di stabilire quali istruzioni eseguire contemporaneamente senza conflitti e quindi capace di raggrupparle in pacchetti d'istruzioni. Così facendo la CPU può limitarsi alla pura gestione dello scheduling di un pacchetto senza inutili ripensamenti.

6. Il livello del Linguaggio Assemblativo

I programmi che si occupano di tradurre in un particolare linguaggio un programma scritto dall'utente in un altro linguaggio sono chiamati traduttori. Il linguaggio nel quale è scritto il programma originale viene chiamato linguaggio sorgente, e linguaggio destinazione quello nel quale viene convertito. Sia il linguaggio sorgente sia il linguaggio destinazione definiscono dei livelli. Se esistesse un processore in grado di eseguire direttamente i programmi scritti nel linguaggio sorgente, tradurre il programma originale nel linguaggio destinazione non sarebbe necessario. Nella traduzione il programma originale non viene eseguito direttamente, ma viene invece convertito in un programma equivalente chiamato programma oggetto o programma eseguibile binario la cui esecuzione è portata avanti solo dopo che la traduzione è stata completata. La traduzione viene realizzata in due passi distinti:

1. generazione di un programma equivalente nel linguaggio destinazione;
2. esecuzione del programma appena generato.

Questi due passi non sono simultanei: il secondo non inizia finché il primo non sia stato completato. Al contrario, nell'interpretazione esiste un unico passo: l'esecuzione del programma sorgente originale.

6.a Introduzione al linguaggio assemblativo

Quando il linguaggio sorgente è essenzialmente una rappresentazione simbolica di un linguaggio macchina numerico, il traduttore è chiamato assemblatore e il linguaggio sorgente è chiamato linguaggio assemblativo. Quando il linguaggio sorgente è un linguaggio ad alto livello come Java oppure C e il linguaggio destinazione è un linguaggio macchina numerico oppure una sua rappresentazione simbolica: in questo caso il traduttore viene chiamato compilatore.

Cos'è un linguaggio assemblativo

Simboli mnemonici (ADD, SUB, ...) sono più semplici da ricordare rispetto a una sequenza di bit 011000... Il linguaggio assemblativo ha una corrispondenza uno ad uno rispetto a quello macchina. Il programmatore assembler ha accesso a tutte le risorse della macchina differentemente da uno che lavora con linguaggi di altro livello. Un

programma in linguaggio assemblativo è scritto per una specifica famiglia di macchina, mentre un programma ad alto livello è “machine independent”.

Perché usare il linguaggio assemblativo

Programmare in Assembler è complesso. Per alcune categorie di applicazioni ove servono ridotte dimensioni del programma, velocità e completo sfruttamento della macchina non esiste alternativa.

Formato delle istruzioni

La struttura delle istruzioni di un linguaggio assemblativo rispecchia la struttura delle istruzioni macchina, delle quali le prime forniscono una rappresentazione simbolica; ciononostante i linguaggi assemblativi di macchine diverse e di livelli differenti hanno un numero sufficiente di somiglianze da permettere di parlare di linguaggio assemblativo in termini generali.

Etichetta	Opcode	Operandi	Commenti
FORMULA:	MOV	EAX,I	; EAX = I
	ADD	EAX,J	; EAX = I + J
	MOV	N,EAX	; N = I + J
I	DD	3	; I=3 (I occupa 4 byte)
J	DD	4	; J=4 (J occupa 4 byte)
N	DD	0	; N=0 (N occupa 4 byte)

Define Double (nell'8088 la parola era a 16 bit!)

Le istruzioni del linguaggio assemblativo sono composte da quattro parti: un campo etichetta, un campo per l'operazione (codice operativo o opcode), un campo per gli operandi e un campo per i commenti. Le etichette sono utilizzate per fornire nomi simbolici agli indirizzi, e sono necessarie per definire le destinazioni alle quali portano le istruzioni che effettuano salti e per poter accedere alle parole di dati memorizzate mediante nomi simbolici. Se un'istruzione è etichettata, l'etichetta inizia (di solito) nella colonna 1.

Pseudoistruzioni

I comandi diretti all'assemblatore sono chiamati pseudoistruzioni o anche direttive dell'assemblatore.

SEGMENT	Indica l'inizio di un segmento (testo,dati,...) con certi attributi.
ENDS	Indica la fine del segmento corrente.
ALIGN	Controlla l'allineamento della prossima istruzione o dei successivi dati.
EQU	Definisce un nuovo simbolo assegnandogli una data espressione.
DB	Allocata spazio per uno o più byte, inizializzandoli.
DW	Allocata spazio per uno o più word (16-bit), inizializzandoli.
DD	Allocata spazio per uno o più doubleword (32-bit), inizializzandoli.
DQ	Allocata spazio per uno o più quadword (64-bit), inizializzandoli.
PROC	Indica l'inizio di una procedura.
ENDP	Indica la fine di una procedura.
MACRO	Indica l'inizio della definizione di macro.
ENDM	Indica la fine della definizione di macro.
PUBLIC	Esporta un nome di identificatore definito nel modulo corrente.
EXTERN	Importa un nome di identificatore da un altro modulo.
INCLUDE	Preleva ed include un altro file.
IF	Inizio assemblaggio condizionale: se la condizione è vera assembra le istruzioni che seguono.
ELSE	Se la condizione IF è falsa assembra le istruzioni che seguono.
ENDIF	Fine assemblaggio condizionale
COMMENT	Definisce un nuovo simbolo per il commento.
PAGE	Genera una interruzione di pagina nel listato.
END	Indica il termine del programma assemblativo.

6.b Le macroistruzioni

Spesso i programmatori di linguaggi assemblativi hanno bisogno di ripetere più volte all'interno di un programma alcune sequenze d'istruzioni. Un approccio alternativo consiste nel trasformare la sequenza in una procedura e nel richiamarla quando è richiesta. Le macro abbreviazione usuale di macroistruzioni, forniscono una soluzione facile ed efficiente al problema di ripetere un'identica, o quasi, sequenza d'istruzioni.

Definizione, chiamata ed espansione di macro

Una definizione di macro è un modo per assegnare un nome a una porzione di testo. Dopo che una macro è stata definita, il programmatore può scriverne il nome al posto della porzione di programma corrispondente. Una definizione di macro è composta dalle seguenti parti principali:

1. un'intestazione che indica il nome della macro che si sta definendo;
2. il testo che costituisce il corpo della macro;
3. una pseudoistruzione che segna la fine della definizione.

L'uso del nome di una macro come codice operativo viene detto chiamata di macro,

mentre la sua sostituzione con il corpo della macro è chiamata espansione di macro.

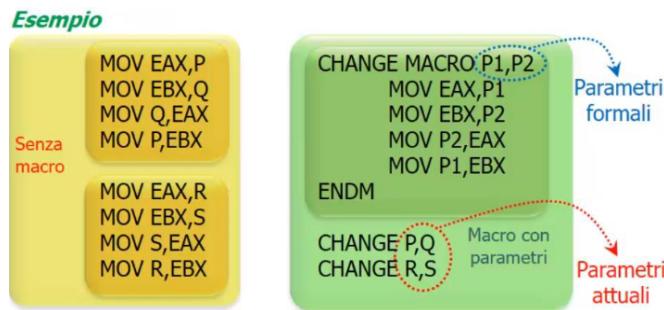
L'espansione della macro viene effettuata durante il processo assemblativo e non

durante l'esecuzione del programma. Le macro non devono essere confuse con le procedure:

Questione	chiamata alla Macro	chiamata alla Procedura
Quando è fatta la chiamata?	Durante l'assemblaggio (compile-time)	durante l'esecuzione (run-time)
Il corpo è inserito nel programma oggetto ogni volta che appare la chiamata?	Sì	No
L'istruzione per la chiamata di procedura è inserita nel programma oggetto e successivamente eseguita?	No	Sì
Occorre utilizzare una istruzione di ritorno dopo aver eseguito una chiamata?	No	Sì
Quante copie del corpo appaiono nel programma oggetto?	Tante quante sono le chiamate	Una

Macro con parametri

La definizione di una macro può includere dei parametri formali che verranno poi sostituiti dai corrispondenti valori attuali. Gli assemblatori di macro gestiscono queste situazioni consentendo che le definizioni delle macro specifichino dei parametri formali e che le chiamate forniscano dei parametri attuali.



Caratteristiche avanzate

Un problema che sorge usando gli assemblatori che supportano le macro è la duplicazione delle etichette. Se la macro viene chiamata due o più volte, l'etichetta

verrà duplicata causando un errore del linguaggio assemblativo. Una soluzione consiste nell'obbligare il programmatore a fornire tramite parametro una diversa etichetta ogni volta che chiama la macro. La soluzione utilizzata da MASM è invece quella di permettere la dichiarazione di etichetta LOCAL, e lasciare che l'assemblatore generi un'etichetta diversa per ogni espansione della macro.

Implementazione delle funzionalità macro negli assemblatori

Per implementare una funzionalità macro un assemblatore deve essere in grado di eseguire due funzioni: salvare le definizioni delle macro ed espandere le loro chiamate.

6.c Il processo di assemblaggio

Assemblatori a due passate

Poiché un programma si compone di istruzioni che possono avere dei "salti" in avanti l'assemblatore non può conoscere in anticipo la posizione dell'istruzione richiamata. Esistono due soluzioni:

- Leggere il programma sorgente due volte costruendo la prima volta una tabella dei simboli, etichette e istruzioni
- Leggere il programma sorgente una volta e convertirlo in un formato intermedio in una tabella in memoria eseguendo poi un secondo passaggio sulla tabella.

In entrambe le soluzioni il primo passaggio ha il compito di espandere tutte le macro.

Prima passata

Il primo passaggio costruisce la tabella dei simboli. Durante questa fase l'assembler utilizza una variabile ILC che memorizza l'indirizzo dell'istruzione che sta assemblando. La maggior parte degli assembler utilizza tre tabelle interne per memorizzare: i simboli, le pseudoistruzioni e i codici operativi.

Etichetta	Opcode	Operandi	Commenti	Lung.	+	ILC
MARIA	MOV	EAX,I	;EAX = I	5		100
	MOV	EBX,J	;EBX = J	6		105
ROBERTA:	MOV	ECX,K	;ECX = K	6		111
	IMUL	EAX, EAX	;EAX = I*I	2		117
	IMUL	EBX, EBX	;EBX = J*J	3		119
MARILYN:	IMUL	ECX, ECX	;ECX = K*K	3		122
	ADD	EAX, EBX	;EAX = I*I+J*J	2		125
	ADD	EAX, ECX	;EAX = I*I+J*J+K*K	2		127
STEPHANY:	JMP	DONE	;branch to DONE	5		129

Symbol	Value
MARIA	100
ROBERTA	111
MARILYN	125
STEPHANY	129

Seconda passata

Durante il secondo passaggio è generato il codice oggetto. Deve generare le informazioni utili al linker per collegare in un unico file eseguibile tutte le procedure assemblate in momenti distinti.

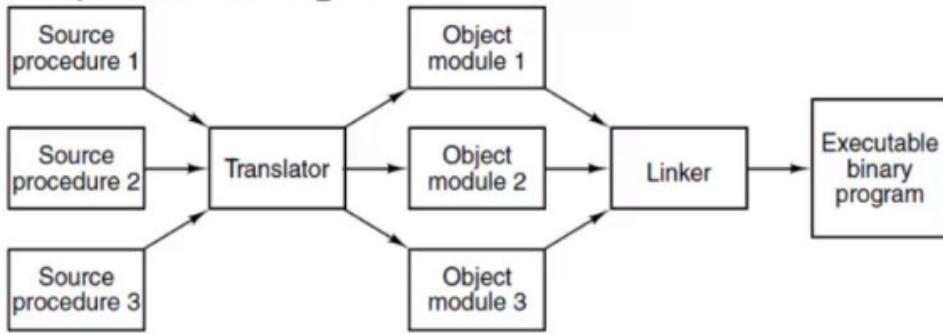
Tabella dei simboli

La tabella dei simboli è una tabella associativa: un insieme di coppie <simbolo, valore> accessibili tramite il simbolo. Esistono varie tecniche per realizzarla:

1. Utilizzare una struttura ordinata ed accedervi in modo dicotomico (mantenere ordinata una struttura di dati costa dal punto di vista computazionale).
2. Usare una codifica hash che mappa i simboli nell'intervallo da 0 a k-1. (problema dell'uniforme distribuzione della funzione e delle collisioni).

6.d Collegamento e caricamento

Tutti i programmi hanno più procedure. Gli assembler traducono una procedura alla volta in linguaggio oggetto e salvano il risultato sul disco. Prima di poter eseguire il programma occorre collegare in modo appropriato tutte le procedure oggetto (linking). In assenza di memoria virtuale occorre caricare, attraverso il loader, il programma generato in memoria centrale prima di eseguirlo.

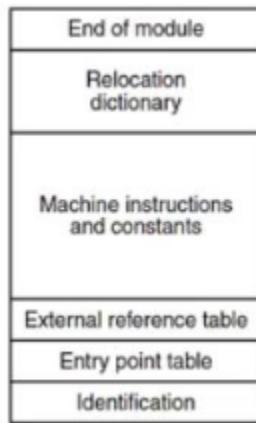


Compiti del linker

La funzione del linker è quella di unire le procedure tradotte separatamente e di collegarle tra loro in modo da poter eseguire come un'unica unità chiamata programma eseguibile binario.

Struttura di un modulo oggetto

Inizialmente troviamo il codice identificativo e le lunghezze delle singole parti del modulo (dati utili al linker). Segue l'insieme dei punti di ingresso a cui possono fare riferimento altri moduli (direttiva PUBLIC). Troviamo poi la lista dei riferimenti utilizzati all'esterno del modulo. A questo punto troviamo il codice assemblato e le costanti (l'unica parte che verrà caricata in memoria al momento dell'esecuzione). Segue il dizionario di rilocazione che fornisce gli indirizzi che dovranno essere rilocati. Infine troviamo l'identificativo di fine modulo, l'indirizzo ove iniziare l'esecuzione, un eventuale checksum per rilevare gli errori che possono avvenire durante la lettura del modulo.



7. Architetture per il calcolo parallelo

L'obiettivo principale dell'industria dei computer è da sempre orientata verso l'incremento delle performance. In passato questo è stato possibile incrementando la frequenza del clock. Ma la velocità dei circuiti non può più aumentare perché si è giunti ad un limite fisico del materiale, perché nessun segnale elettrico può propagarsi più velocemente della velocità della luce. Un modo per ottenere una maggiore potenza di calcolo è attraverso l'utilizzo delle architetture parallele: molte CPU (con velocità normale) che collaborano per il conseguimento del medesimo obiettivo.

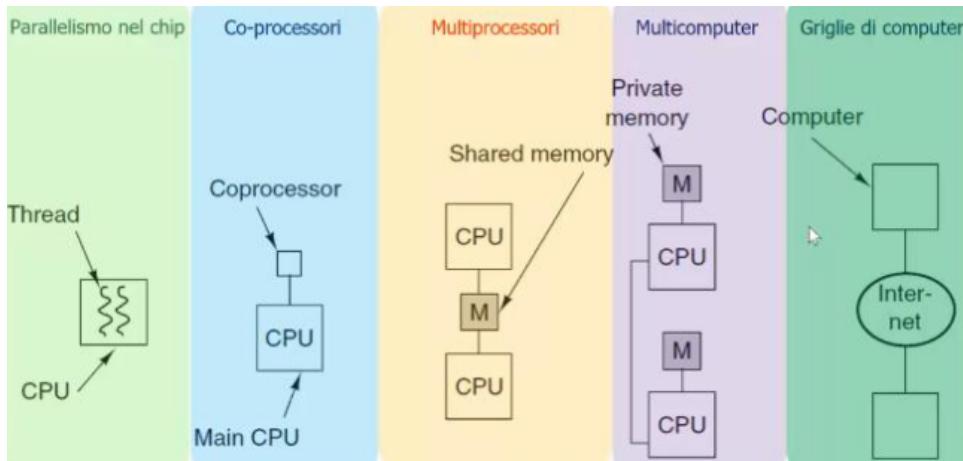
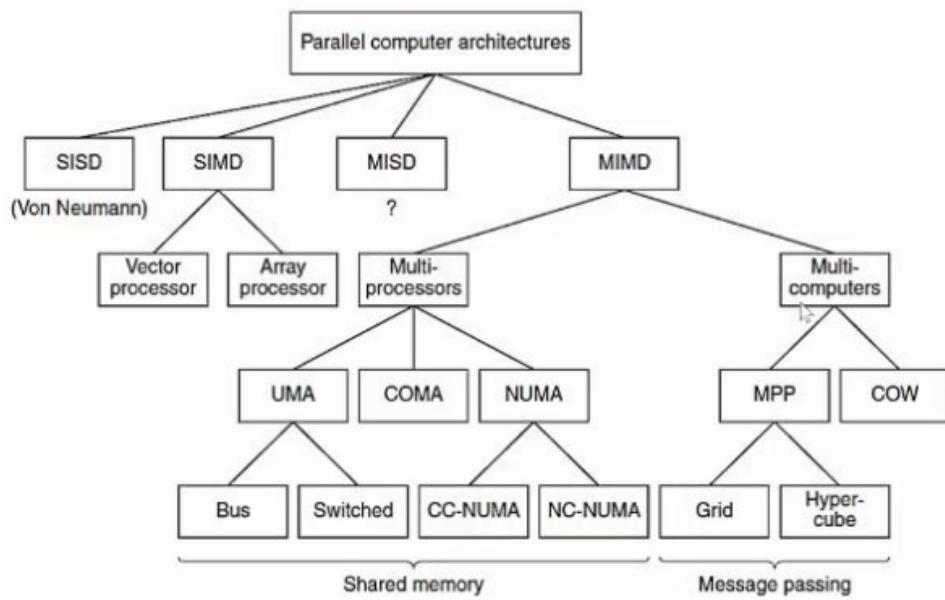
Il parallelismo nel chip della CPU (attraverso un progetto superscalare a pipeline) produce un fattore di miglioramento da 5 a 10.

Per incrementare drasticamente le performance di un calcolatore occorre progettare sistemi con molte CPU.

7.a Classificazione di Flynn

La classificazione si basa su due concetti: il flusso di istruzioni e il flusso di dati

Flusso di Istruzioni	Flusso di Dati	Nome	Esempio
Singolo	Singolo	SISD	Modello di Von Neumann
Singolo	Multiplo	SIMD	Supercomputer vettoriali
Multiplo	Singolo	MISD	Non sono note
Multiplo	Multiplo	MIMD	Multiprocessori e Multicomputer



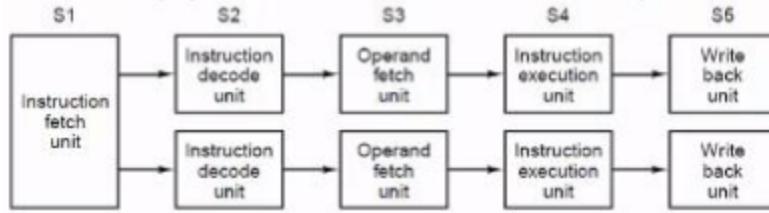
7.b Parallelismo dei chip

Obiettivo: far svolgere al chip più compiti alla volta.

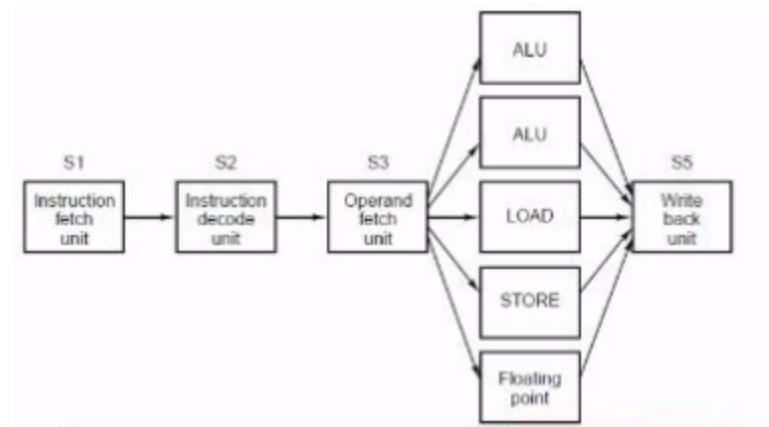
Parallelismo a livello delle istruzioni

L'idea è di emettere più istruzioni per ciclo di clock. Ci sono due tipi di CPU a emissione multipla: processori superscalari e processori VLIW ("Very Long Instruction Word", con parole di istruzione molto lunghe).

Le CPU superscalari sono composte da pipeline e più unità funzionali.



I processori VLIW sono in grado di indirizzare le diverse unità funzionali con una sola linea di pipeline.



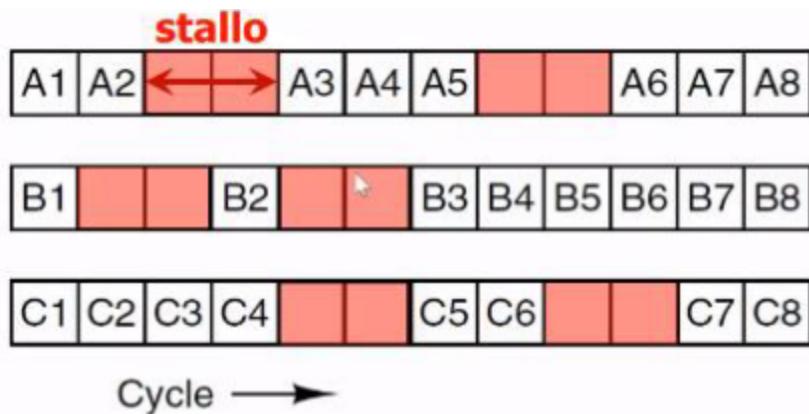
Multithreading nel chip

Tutte le moderne CPU a pipeline presentano un problema: quando una CPU tenta di accedere ad un riferimento in memoria che non è nella cache (cache miss), bisogna aspettare il caricamento nella cache prima di riprendere l'esecuzione e così nel frattempo la pipeline è in stalllo.

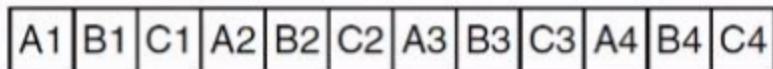
Il multithreading nel chip permette di mascherare queste situazioni attraverso lo switch tra thread. Esistono vari approcci:

Multithreading a grana fine

Supponiamo di avere una CPU capace di emettere un'istruzione per ciclo di clock con tre thread A, B e C. Durante il primo ciclo, A, B e C eseguono le istruzioni A1, B1 e C1. Purtroppo al secondo ciclo di clock A2 fa riferimento non presente nella cache di primo livello e deve attendere 2 cicli per recuperarlo dalla cache di secondo livello.



Il multithreading a grana fine sono eseguite ogni ciclo di clock, a turno le singole istruzioni dei thread nascondendo così gli stalli.



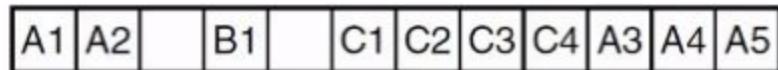
Dal momento che non vi è alcuna relazione tra i thread, ciascuno ha bisogno del proprio insieme di registri. All'emissione di un'istruzione è necessario accludere all'istruzione stessa un puntatore al suo insieme di registri così che l'hardware possa sapere quale registro usare. Per questa ragione, il numero massimo di thread che può essere eseguito in parallelo è stabilito a priori in fase di progettazione del chip.

Le operazioni di memoria non sono l'unica causa di stallo: alcune istruzioni condizionano l'esecuzione di altre.

Nella pipeline non ci sarà mai più di un'istruzione per thread e quindi il numero massimo di thread è pari al numero di stadi della pipeline.

Multithreading a grana grossa

In questo caso un thread va avanti finché non va in stall, causando lo spreco di un ciclo. A quel punto l'esecuzione viene commutata su un altro thread.



Visto che si perde un ciclo a ogni stallo, il multithreading a grana grossa è potenzialmente meno efficiente di quello a grana fine, ma presenta il vantaggio di

richiedere meno thread per mantenere la CPU occupata.

Esiste una variante che permette di “guardare avanti” le istruzioni anticipando lo stallo e approssimando il multithreading a grana fine.

Indipendentemente dal tipo di multithreading usato, è necessario mantenere traccia dell'appartenenza delle operazioni ai thread. Nel caso del multithreading a grana fine l'unica possibilità ragionevole è l'inserimento di un identificatore di thread in ogni operazione, così da poter rintracciare la sua identità mentre attraversa i diversi stadi della pipeline. Nel caso del multithreading a grana grossa si può far di meglio: svuotare la pipeline a ogni commutazione di thread. In tal modo c'è sempre un solo thread nella pipeline e così la sua identità non è mai messa in dubbio. Questa soluzione ha senso solo se il tempo che intercorre tra due commutazioni è molto più lungo del tempo di svuotamento della pipeline.

Fin qui abbiamo presupposto che la CPU possa emettere una sola istruzione per ciclo, ma abbiamo già visto che le CPU moderne ne possono emettere di più. Supponiamo che la CPU possa emettere due istruzioni per ciclo, pur conservando la regola che, se un'istruzione va in stallo, le successive non possono essere emesse. Praticamente, i thread si succedono a turno e la CPU emette due istruzioni per ogni thread finché non incontra uno stallo, nel qual caso commuta al thread successivo all'inizio del ciclo seguente.

A1	B1	C1	A3	B2	C3	A5	B3	C5	A6	B5	C7
A2		C2	A4		C4		B4	C6	A7	B6	C8

Multithreading a Grana fine

A1	B1	C1	C3	A3	A5	B2	C5	A6	A8	B3	B5
A2		C2	C4	A4			C6	A7		B4	B6

Multithreading a Grana grossa

Multithreading simultaneo

C'è una terza possibilità di multithreading con le CPU superscalari, il multithreading simultaneo.

Ogni thread emette due istruzioni per ciclo fintanto non raggiunge lo stallo: viene emessa immediatamente un'istruzione del thread che segue affinché la CPU resti pienamente impegnata. Il multithreading simultaneo aiuta anche a mantenere occupate le unità funzionali.

A1	B1	C2	C4	A4	B2	C6	A7	B3	B5	B7	C7
A2	C1	C3	A3	A5	C5	A6	A8	B4	B6	B8	C8

Multiprocessori in un solo chip

Con l'andare avanti della tecnologia, i transistor sono diventati più piccoli ed è stato possibile incrementare il loro numero in un singolo chip. A causa del raggiungimento dei limiti fisici dei materiali, come la dissipazione del calore, non è possibile incrementare la frequenza del clock per ottenere CPU più veloci.

Perciò si inseriscono più CPU (chiamate core) all'interno dello stesso chip (o meglio die).

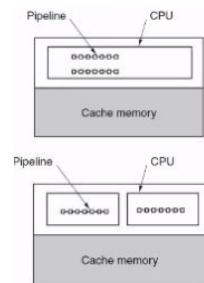
I multiprocessori possono essere realizzati con core identici (multiprocessori omogenei) oppure con core con specifiche funzionalità (multiprocessori eterogenei).

Multiprocessori omogenei in un solo chip

Le CPU che condividono le cache, e la memoria principale sono dette multiprocessori. Esistono due tecnologie in un solo chip.

Quelle con una sola CPU e più pipeline che possono moltiplicare il throughput in base al numero di pipeline. Le unità funzionali sono intimamente correlate.

Quelle che hanno più core ciascuno con la propria pipeline. L'interazione tra core non è semplice poiché risultano maggiormente disaccoppiate.



Mentre le CPU possono o meno condividere le cache, esse condividono sempre la memoria principale. Il processo di snooping (eseguito dall'hardware) garantisce che se una parola è presente in più cache e una CPU modifica il suo valore in memoria, essa è automaticamente rimossa in tutte le cache in modo da garantire la consistenza.

Multiprocessori eterogenei in un solo chip

In questo caso ogni core ha un compito specifico (come ad esempio decoder audio/video, crittoprocessore, interfacce di rete). Poiché queste architetture realizzano

un vero e proprio calcolatore completo in un singolo chip sono spesso dette system on a chip.

Come accaduto nel passato, l'hardware è molto più avanti del software: mentre sono attualmente disponibili dei chip multicore, non abbiamo applicazioni in grado di sfruttare queste nuove caratteristiche.

Pochi programmati sono in grado di scrivere algoritmi paralleli che gestiscano correttamente la competizione delle risorse condivise.

I chip multicore sono come dei piccoli multiprocessori per questa ragione vengono chiamati CMP (Chip-level MultiProcessor ovvero multiprocessori a livello di chip)

Dal punto di vista software essi non sono coi differenti dai multiprocessori a bus o a reti di switch.

Rispetto a multiprocessori a bus che hanno una cache per ogni CPU, potrebbero avere delle prestazioni degradate sulla cache condivisa nelle situazioni in cui un core saturi la cache L2.

Altra differenza rispetto ai multiprocessori è la minore tolleranza ai malfunzionamenti causata dalla stretta connessione dei core (un errore potrebbe propagarsi negli altri).

7.c Coprocessori

Per velocizzare un calcolatore si può aggiungere un secondo processore dedicato a specifiche funzioni (coprocessore). Esistono molte varianti di coprocessori:

- Processori di rete.
- Processori grafici.
- Crittoprocessori.

In alcuni casi la CPU assegna al coprocessore un'istruzione o un insieme d'istruzioni e gli ordina di eseguirle; in altri casi il coprocessore è più indipendente e lavora per conto proprio. In tutti i casi, ciò che li caratterizza è il fatto di assistere nell'esecuzione un altro processore, che resta il processore principale.

Elaborazione di rete(processori di rete), multimedia (processori grafici) e crittografia (crittoprocessori).

7.d Multiprocessori

Il passo successivo è la combinazione di più CPU per formare sistemi più grandi. Questi sistemi si dividono in due categorie: multiprocessori e multicomputer.

Multiprocessori e multicomputer a confronto (MIMD)

Tutte le comunicazioni tra componenti elettronici e ottici avvengono attraverso lo scambio di messaggi.

Le principali differenze tra MIMD risiedono nella base dei tempi, nella scala delle distanze e nell'organizzazione logica utilizzata. Troviamo:

1. Multiprocessori a memoria condivisa, meno di 1000 CPU comunicano attraverso una memoria condivisa, il tempo di accesso ad una parola di memoria è di 2-10 ns.
2. Multicomputer a scambio di messaggi, un certo numero di coppie memoria-CPU sono collegati attraverso una rete di interconnessione ad alta velocità; le comunicazioni avvengono attraverso brevi messaggi che possono essere spediti in 10-50 ns; ogni memoria è locale a ciascuna CPU.
3. Sistemi distribuiti, computer distribuiti su un'area geografica estesa come internet, i messaggi impiegano da 10 a 100 ms.

Multiprocessori

Un multiprocessore a memoria condivisa (o semplicemente multiprocessore) è un calcolatore in cui tutte le CPU condividono una memoria comune.

L'unica proprietà inusuale che ha un multiprocessore è che una CPU può scrivere un valore in una parola di memoria e trovarvi un differente valore alla successiva lettura (perché un'altra CPU può averla modificata). I sistemi operativi utilizzati nei multiprocessori sono normali sistemi operativi.

Esistono due differenti tipologie di multiprocessori:

- Multiprocessori UMA (Uniform Memory Access) in cui ogni parola di memoria può essere letta alla stessa velocità.
- Multiprocessori NUMA (NonUniform Memory Access) in cui non tutte le parole di memoria possono essere lette alla medesima velocità.

UMA

UMA con singolo bus

I più semplici multiprocessori sono basati su un singolo bus che interconnette tutte le CPU alla memoria condivisa.

Una CPU che vuole leggere/scrivere una parola in memoria, se il bus è occupato, deve attendere che si liberi. Questa architettura funziona bene con due o tre CPU (il contenzioso sull'accesso al bus può essere gestito agevolmente). Per poter incrementare il numero di CPU, e quindi di performance, occorre aggiungere delle ulteriori memorie.

UMA con singolo bus e cache nelle CPU

L'aggiunta di una memoria cache all'interno delle singole CPU può ridurre il traffico sul bus. Il Caching non viene eseguito sulle singole parole di memoria ma su blocchi di 32 o 64 byte. Quando una parola è referenziata, l'intero blocco che la contiene chiamato, linea di cache, è caricato nella CPU che l'ha richiesta. Ogni blocco di cache è contrassegnato come Read-Only o Read-Write.

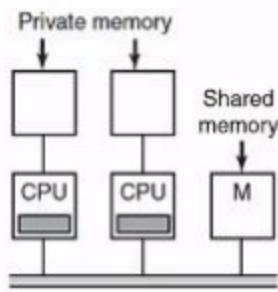
Se una CPU scrive una parola che è contenuta anche in altri blocchi di cache remote, l'hardware del bus percepisce la scrittura e lo segnala alle altre cache che possono avere:

- Una copia "pulita" del blocco di memoria, allora la cache la sovrascrive con il valore aggiornato
- Una copia modificata del blocco di memoria, allora la cache lo deve prima trascrivere in memoria e poi applicare la modifica.

L'insieme di tali regole, utilizzate per la gestione della cache, è chiamato protocollo di coerenza della cache.

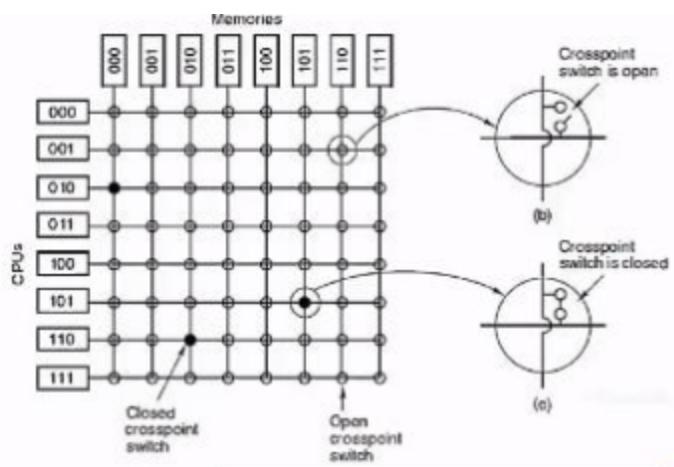
UMA con singolo bus e CPU dotate di RAM

Con l'aggiunta di memorie RAM in un bus interno dedicato per ogni CPU la memoria condivisa è utilizzata esclusivamente per scrivere variabili condivise (globali). Questa soluzione riduce il traffico sul bus ma richiede una collaborazione attiva del compilatore che deve separare gli oggetti locali (programma, stack, variabili locali, ...) da quelle globali.



UMA con crossbar switch

Anche con le migliori tecniche di caching, l'uso di un solo bus di interconnessione limita la dimensione dei multiprocessori UMA a circa 16 o 32 CPU. Per superare questo limite occorre utilizzare un diverso tipo di rete di interconnessione. Il circuito più semplice che collega n CPU a k memorie è il crossbar switch.



A ogni intersezione tra una linea orizzontale (CPU) e una verticale (RAM) c'è un crosspoint ("punto di incrocio"). Un crosspoint è un piccolo interruttore che permette di stabilire il collegamento tra CPU e RAM.

Attraverso questo sistema si realizza una rete non bloccante: ad alcuna CPU è mai vietata la connessione verso una memoria di cui ha bisogno perché la linea è già occupata.

Inoltre non c'è bisogno di alcuna pianificazione anticipata. è sempre possibile collegare una CPU alla memoria aprendo o chiudendo un interruttore.

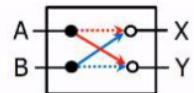
Rimane il problema della competizione per la memoria, qualora due, o più, CPU vogliono accedere allo stesso modulo nel medesimo istante (partizionando la memoria

in n unità, la competizione si riduce di un fattore n rispetto al modello con bus singolo).

Una delle peggiori caratteristiche di questo schema è che il numero degli incroci cresce come n^2 . Con 1000 CPU e moduli di memoria occorrono un milione di crosspoints. Costruire un crossbar di queste dimensioni non è fattibile.

UMA con rete di commutazione multilivello

Una progettazione di multiprocessori differente è basata su commutatore 2x2 (due input e output). I messaggi che arrivano nei due input possono scambiarsi in una due uscite.

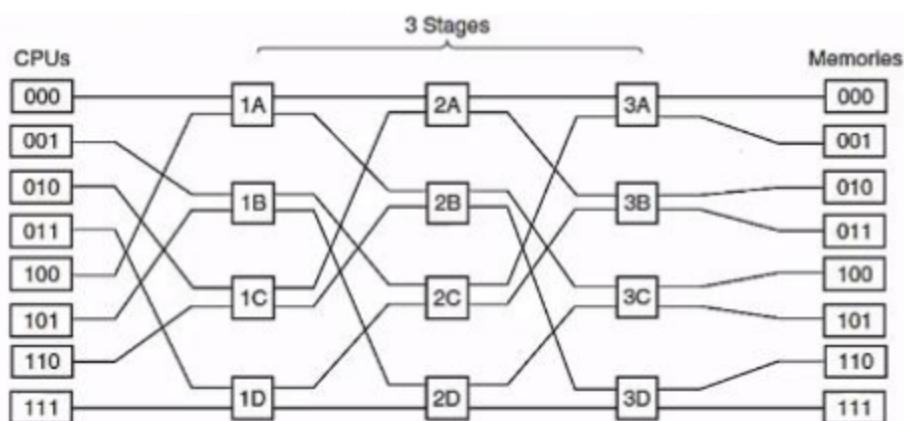


Ogni messaggio contiene:

- Modulo: quale memoria usare;
- Indirizz: specifica l'indirizzo nel modulo;
- Opcode: il codice di operazione da eseguire;
- Valore: il valore dell'operando.

Lo switch utilizza il campo Modulo e lo usa per scegliere dove spedire il messaggio (su X o Y). Gli switch possono essere organizzati in molti modi per costruire reti di commutazione multilivello.

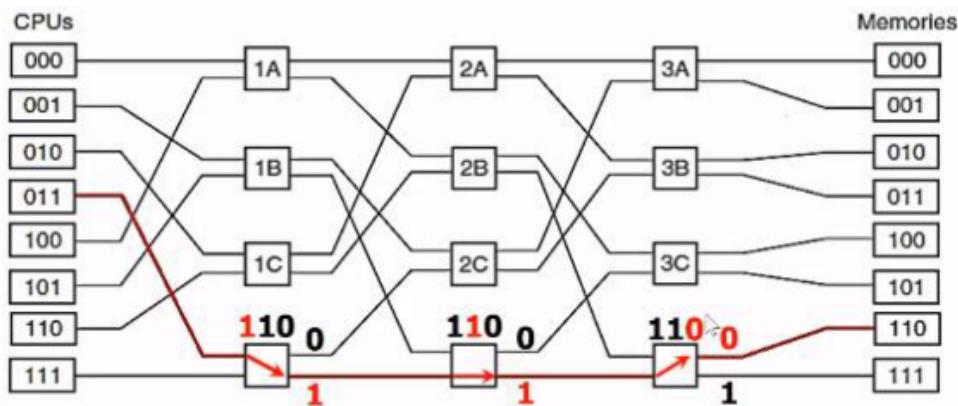
Una possibilità economica e semplice è la rete omega.



In questo caso ci sono 8 CPU connesse a 8 memorie tramite 12 commutatori (switch). Per n CPU e n memorie ci vogliono $\log_2 n$ livelli con $n/2$ commutatori per livello, per un totale di $(n/2)\log_2 n$ commutatori: è un numero molto inferiore a n^2 , specie per valori grandi di n .

Lo schema di connessione di una rete omega è detto shuffle (o miscuglio) perfetto.

Per capire il funzionamento di una rete omega, supponiamo che la CPU 3 (011) voglia leggere una parola dal modulo di memoria 6 (110).



Quindi, a differenza dell'interconnessione con crossbar switch la rete omega è una rete bloccante: non tutti gli insieme di richieste possono essere processati contemporaneamente.

È auspicabile distribuire i riferimenti alla memoria in modo uniforme rispetto ai moduli. Una sistema di memoria in cui le parole consecutive sono in moduli diversi è detto interleaved. Le memorie interleaved massimizzano il parallelismo perché la maggior parte dei riferimenti a memoria è in indirizzi consecutivi.

È sempre possibile progettare reti di switch non bloccanti.

NUMA

Con i processori UMA a singolo bus si può arrivare a connettere fino a 12 CPU, con un rete di interconnessione, a cause dei costi dell'hardware, si può arrivare a connettere insieme meno di 100 CPU.

Per eccedere questo valore occorre introdurre una nuova architettura in cui si rinuncia a qualcosa: che i tempi di accessi della memoria siano uniformi. In un processore NUMA il tempo di accesso ai moduli di memoria locale è minore rispetto a quello dei moduli remoti.

Tutti i programmi che girano su multiprocessori UMA continuano a girare sulle macchine NUMA, ma naturalmente hanno performance degradate

Le macchine NUMA hanno tre caratteristiche chiavi:

- C'è un unico spazio di indirizzamento visibile a tutte le CPU;
- L'accesso alla memoria remota è attraverso istruzioni di LOAD e STORE;
- L'accesso alla memoria remota è più lento dell'accesso rispetto alla memoria locale.

Ci sono due tipi di macchine NUMA:

- No Cache NonUniform Memory Access (NC-NUMA): quando il tempo di accesso alla memoria distante non è nascosto (perché non c'è caching);
- Cache Coherent NonUniform Memory Access (CC-NUMA): quando sono presenti cache coerenti.

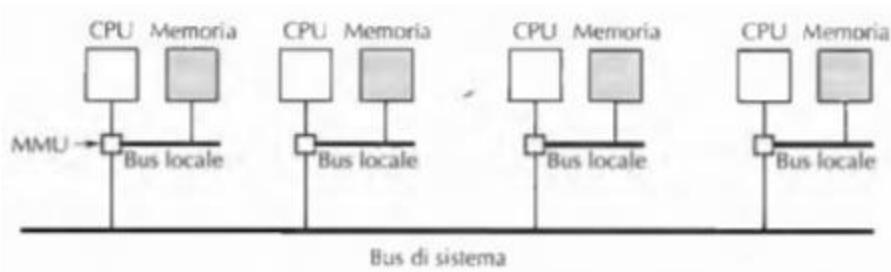
Multiprocessori NC-NUMA

Una macchina NC-NUMA contiene varie CPU, ciascuna dotata di una piccola memoria cui accedeva tramite un bus locale. Inoltre, le CPU erano collegate attraverso un bus di sistema.

Quando la MMU (opportunamente modificata) riceveva una richiesta di accesso a memoria, stabiliva per prima cosa se la parola si trovava in memoria locale. In tal caso la richiesta veniva inoltrata lungo il bus locale per ottenere la parola. In caso contrario, la richiesta veniva instradata lungo il bus di sistema verso il processore che conteneva la parola, che rispondeva di conseguenza. Naturalmente questa seconda evenienza impiegava molto più tempo della prima.

La coerenza della memoria in una macchina NC-NUMA è garantita dal fatto che non c'è caching. Ogni parola di memoria si può trovare in una sola locazione, perciò non c'è pericolo che ne esista una copia difforme: non ci sono copie.

D'altro canto, la collocazione di una pagina in memoria diventa molto importante perché il degrado delle prestazioni dovuto a un cattivo posizionamento dei dati è molto rilevante.



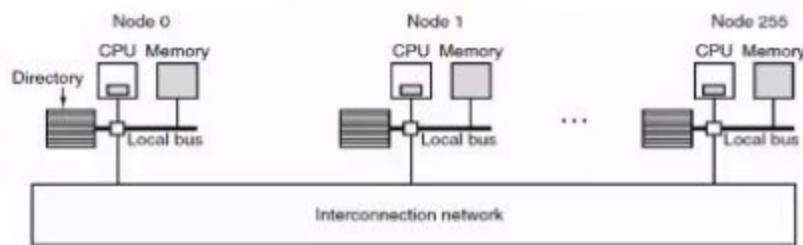
Multiprocessori CC-NUMA

Dover accedere alla memoria remota ogni volta che si fa riferimento a una parola di memoria non locale vuol dire pagare un alto prezzo in termini di prestazioni. Se però si aggiunge il caching, bisogna assicurare anche la coerenza delle cache. Un modo possibile per garantirla è permettere lo snooping del bus di sistema.

L'approccio più comune utilizzato per costruire grandi multiprocessori CC-NUMA è il multiprocessore basato sulle directory (directory-based multiprocessor).

L'idea di fondo è di mantenere una base dati delle linee di memoria ed il loro stato. Quando è referenziata una linea di cache, viene interrogata la base dati per sapere dove si trova e se è pulita o sporca.

Poiché ogni istruzione che accede alla memoria interroga questo database, occorre che sia gestito su un hardware specifico velocissimo. L'unione di tutte le memorie dei nodi costituisce l'intero spazio di indirizzamento.



7.e Tipi di sistema operativo multiprocessore

Cenni di memoria virtuale

All'inizio dell'era informatica, le memorie dei computer erano costose e poco capienti. In quel periodo, i programmati impiegavano molto tempo nel tentativo di "assottigliare" i programmi perché entrassero in memorie minuscole.

La soluzione tradizionale a questo problema era l'uso di una memoria secondaria, per esempio di un disco. Il programmatore divideva il programma in un certo numero di pezzi, detti overlay ("ricoprimento"), ciascuno dei quali poteva entrare in memoria. Al fine di eseguire il programma si recuperava il primo overlay e lo si eseguiva per un po'. Al termine della sua esecuzione il primo pezzo leggeva l'overlay successivo e lo mandava in esecuzione, e così via.

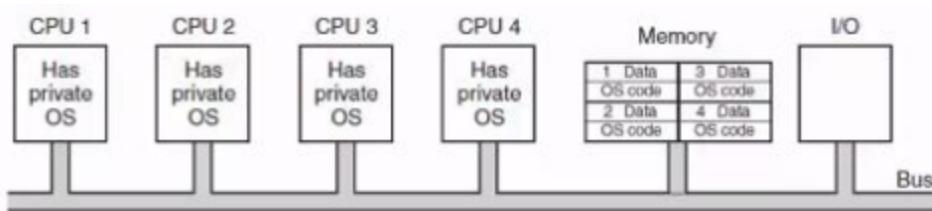
Nel 1961 si trovò un metodo per eseguire il processo di overlay automaticamente, in modo tale che il programmatore non avrebbe dovuto neanche accorgersi della sua presenza.

Questo metodo, oggi chiamato memoria virtuale, presentava il vantaggio evidente di liberare il programmatore da un grosso lavoro di noiosa preparazione.

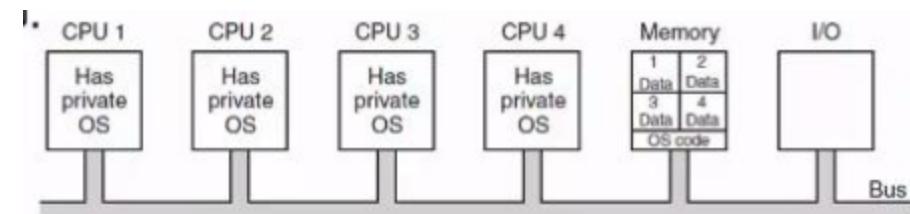
Ci sono tre differenti approcci per i sistemi operativi multiprocessori:

Ogni CPU ha il proprio sistema operativo

Il modo più semplice per organizzare un sistema operativo multiprocessore consiste nel dividere staticamente la memoria in tante partizioni quante sono le CPU. Ogni CPU ha un proprio sistema operativo e una propria memoria privata.



Le CPU operano come computer indipendenti, una possibile ottimizzazione è attraverso la condivisione del sistema operativo tra CPU e fare copie private delle sole strutture dati del sistema operativo.



Questo schema è migliore rispetto a n computer separati, poiché permette alle CPU di condividere un insieme di dispositivi di I/O. Però, presenta diverse problematiche:

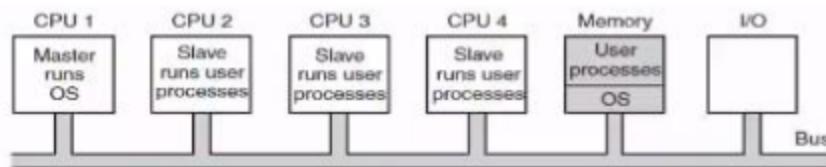
1. Quando un processo effettua una chiamata di sistema, essa è gestita dalla CPU che l'ha lanciato, utilizzando le strutture dati presenti nelle tabelle del proprio sistema operativo;
2. Non c'è condivisione di processi, ogni CPU gestisce indipendentemente i propri processi. Quindi non è possibile il bilanciamento del carico (una CPU potrebbe

essere scarica mentre un'altra satura);

3. Non c'è condivisione di pagine. Una CPU può eseguire molto swap a causa della carenza di spazio libero mentre un'altra può disporre di pagine libere;
4. Se un sistema operativo mantiene un buffer cache dei blocchi del disco usati recentemente indipendentemente dagli altri questo può condurre a letture inconsistenti (cause da letture sporche nei vari buffer).

Multiprocessori Master-Slave

Il sistema operativo e le sue tabelle sono presenti sulla CPU 1 (master), tutte le chiamate di sistema sono reindirizzate alla CPU 1 per essere processate.



Questo modello risolve la maggior parte dei problemi del modello precedente:

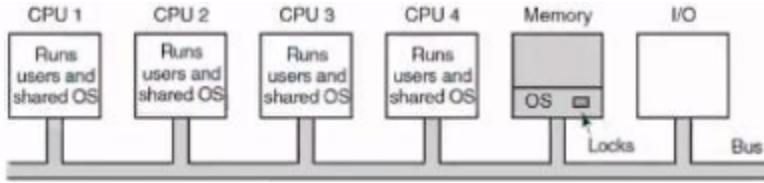
- C'è una singola struttura dati che tiene traccia dei processi pronti;
- La CPU master può bilanciare il carico sulle varie CPU;
- Le pagine sono allocate dinamicamente tra i processori e c'è una sola buffer cache che evita le inconsistenze.

Il problema è che, con molte CPU, il master deve gestire tutte le chiamate di sistema. Questo modello è utilizzabile per sistemi con poche CPU.

Multiprocessori simmetrici

L'SMP (MultiProcessore Simmetrico) elimina l'asimmetria del modello master-slave: in pratica, ogni CPU può diventare master.

Una copia del sistema operativo è in memoria e ciascuna CPU può eseguirla. Quando si fa una chiamata di sistema, La CPU riceve la chiamata effettua una trap al kernel ed elabora la richiesta.



Questo schema bilancia processi e memoria dinamicamente (visto che c'è solo un insieme di tabelle del sistema operativo) ed elimina il collo di bottiglia.

D'altra parte introduce le seguenti problematiche:

- Le CPU devono essere sincronizzate a causa della condivisione del sistema operativo;
- Bisogna evitare i deadlock (potrebbe essere impossibile risolverli);
- La gestione è critica perché dipende dall'esperienza del programmatore nel contesto del sistema.

Sincronizzazione dei multiprocessori

Le CPU in un multiprocessore hanno bisogno di sincronizzarsi: le regioni critiche del kernel e le tabelle devono essere protette da mutex.

Se si disabilitano gli interrupt di una CPU, le altre CPU continuano a funzionare normalmente e possono accedere ad un'area critica.

In un mono processore l'istruzione TSL impiegata nei mutex, permette ad una parola di memoria di essere controllata e imposta in una sola operazione indivisibile.

Con più CPU, l'istruzione TSL deve prima bloccare il bus evitando l'accesso di altre CPU, poi accedere alla memoria e quindi sbloccare il bus.

Se la TSL è realizzata correttamente, la CPU che fa richiesta del bus cerca di verificarne la disponibilità in un ciclo rapidissimo in modo da non rallentare le altre CPU (spin lock).

Un approccio alternativo è lo switch dei processi, una CPU invece che attendere (spinning) passa il controllo ad un altro processo.

Non esiste la soluzione ideale, un approccio ibrido potrebbe essere di attendere per un certo tempo e poi fare un scambio di processo oppure memorizzare i tempi medi di attesa e vedere se il tempo attuale rispetta i tempi medi o è fuori norma (a meno di un certo errore).

Schedulazione dei multiprocessori

Su un monoprocessoresso lo scheduling è mono-dimensionale. La sola domanda cui si deve rispondere è: "qual è il prossimo processo da eseguire?".

In un multiprocessoresso, lo scheduling è bidimensionale: lo schedulatore deve decidere quale processo eseguire, e su quale CPU eseguirlo.

Un altro fattore di complicazione è che in alcuni sistemi non tutti i thread sono correlati o lo sono in gruppi.

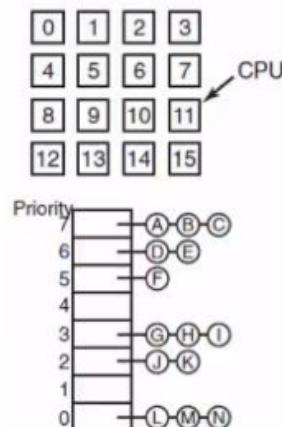
Esistono vari algoritmi di scheduling per ciascuna situazione:

Timesharing

Lo scheduling a condivisione di tempo (timesharing) è utilizzato quando i thread sono indipendenti.

L'algoritmo utilizza un vettore di liste di thread (tutti nello stato "ready") a diverse priorità di esecuzione).

Il fatto di avere un'unica struttura dati utilizzata da tutte le CPU ripartisce il tempo tra le CPU come se fossero in un sistema monoprocessoresso, fornendo anche un sistema automatico di bilanciamento del carico poiché non accade mai che una CPU è inattiva e un'altra sovraccarica.



Questo algoritmo presenta due potenziali problemi:

- Le dispute per l'accesso alla struttura dati di scheduling con il crescere del numero di CPU;
- Il normale sovraccarico che si presenta per fare lo scambio di contesto quando un processo si blocca per l'I/O.

Supponiamo che un thread termini il suo quantum di tempo mentre mantiene uno spin lock, le altre CPU dovranno attendere il rilascio della risorsa che avverrà quando il processo tornerà attivo e rilascerà il lock. In alcuni sistemi, quando ciò accade, il thread che esegue un lock attiva un flag speciale di processo che permette allo scheduler di non bloccare il processo ma di concedergli un extra-quantum (smart scheduling).

Alcuni multiprocessoressi utilizzano lo scheduling per affinità (affinity scheduling), cioè tentano di far eseguire lo stesso thread alla medesima CPU. La CPU che potrebbe

avere nella propria cache dei blocchi già caricati e questo migliora di molto le performance.

Per creare questa affinità si utilizza un algoritmo di schedulazione a due livelli:

- Quando è creato un thread viene assegnato alla CPU più scarica (top-level scheduling);
- Lo scheduling reale è fatto separatamente da ogni CPU (bottom-level scheduling) utilizzando l'affinità della cache e le priorità dei thread.

Lo scheduling a due livelli presenta tre vantaggi:

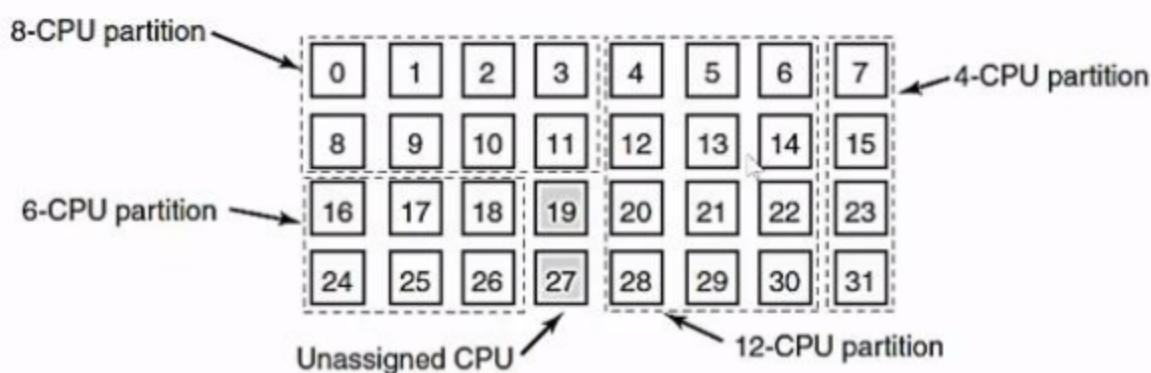
- Distribuisce il carico in modo equo fra le CPU disponibili;
- L'utilizzo delle affinità della cache migliora le performance;
- Si minimizzano le dispute sulle liste di processi poiché ogni CPU tenta di riusare i propri processi.

Condivisione dello spazio

Se accade che i thread sono interrelati si può utilizzare lo scheduling a condivisione di spazio (space sharing). Lo scheduling di molti processi nello stesso tempo su diverse CPU è chiamato space sharing scheduling.

Supponendo di voler creare un intero gruppo di thread correlati, in un solo colpo; lo scheduler controlla se ci sono tante CPU libere quanti sono i thread. Se esistono, ad ogni thread è assegnata una CPU dedicata ed è avviato; al contrario, nessun processo è avviato fintantoché non ci sono disponibili un numero adeguato di CPU.

Ad ogni istante di tempo, l'insieme delle CPU è partizionato staticamente, ciascun gruppo delle quali esegue il gruppo di processi correlati.



Un vantaggio evidente dello scheduling a condivisione di spazio è l'eliminazione della multiprogrammazione che elimina anche l'overhead dovuto agli scambi di contesto.

Uno svantaggio è il tempo sprecato quando si blocca una CPU perché non c'è nulla da fare.

Schedulazione gang

Conseguentemente, si sono creati algoritmi che tentassero di schedulare spazio e tempo insieme.

Una soluzione è il gang scheduling che si compone di tre passaggi:

- I gruppi di thread correlati sono schedulati come un'unità (o gang);
- Tutti i membri della gang sono in esecuzione simultaneamente, in differenti CPU in timesharing;
- Tutti i thread della gang iniziano e terminano la loro porzione di tempo contemporaneamente.

Il tempo è diviso in quantum distinti e le CPU sono schedulate in sincronia. All'inizio di ciascun nuovo quantum tutte le CPU sono rischedolate. Se un thread si blocca, la sua CPU rimane inattiva fino allo fine del quantum.

		CPU					
		0	1	2	3	4	5
Time slot		A ₀	A ₁	A ₂	A ₃	A ₄	A ₅
	0	B ₀	B ₁	B ₂	C ₀	C ₁	C ₂
	1	D ₀	D ₁	D ₂	D ₃	D ₄	E ₀
	2	E ₁	E ₂	E ₃	E ₄	E ₅	E ₅
	3	A ₀	A ₁	A ₂	A ₃	A ₄	A ₅
	4	B ₀	B ₁	B ₂	C ₀	C ₁	C ₂
	5	D ₀	D ₁	D ₂	D ₃	D ₄	E ₀
	6	E ₁	E ₂	E ₃	E ₄	E ₅	E ₆
	7						

Abbiamo un multiprocessore con sei CPU, usate da cinque processi, da A ad E, per un totale di 24 thread pronti. Durante l'intervallo di tempo 0, sono schedulati ed eseguiti i thread da A₀ ad A₅, durante l'intervallo di tempo 1, i thread B₀, B₁, B₂, C₀, C₁, C₂. Durante l'intervallo 2 sono mandati in esecuzione i cinque thread di D e E₀. I rimanenti

sei thread del processo E sono eseguiti nell'intervallo di tempo 3. Quindi il ciclo si ripete, con l'intervallo 4, identico all'intervallo 0.

L'idea della schedulazione gang consiste nell'avere tutti i thread di un processo in esecuzione insieme, cosicché se uno di essi manda una richiesta ad un altro, quest'ultimo riceverà il messaggio e sarà in grado di rispondere quasi immediatamente. Poiché tutti i thread di A sono in esecuzione insieme durante un quantum, in tale intervallo di tempo possono mandare e ricevere un gran numero di messaggi.

7.f Multicomputer

I multiprocessori offrono un modello semplice di comunicazione: tutte le CPU condividono una memoria comune. I thread possono scrivere messaggi in memoria che poi possono, a loro volta, essere letti da altri thread. La sincronizzazione può essere fatta utilizzando mutex, semafori o monitor.

I multiprocessori di grandi dimensioni sono difficili da costruire e risultano costosi. Per ovviare a questi problemi sono nati i multicomputer che sono fortemente accoppiati ma non condividono memoria. Ogni computer ha una propria memoria. Questi sistemi sono anche chiamati Cluster di Computers o Cluster of Workstations (COW).

I multicomputer sono facili da costruire perché il componente base è un PC con una scheda di rete con alte performance. Il segreto di ottenere alte performance in un multicomputer sta nel progetto della rete di interconnessione e delle schede di rete.

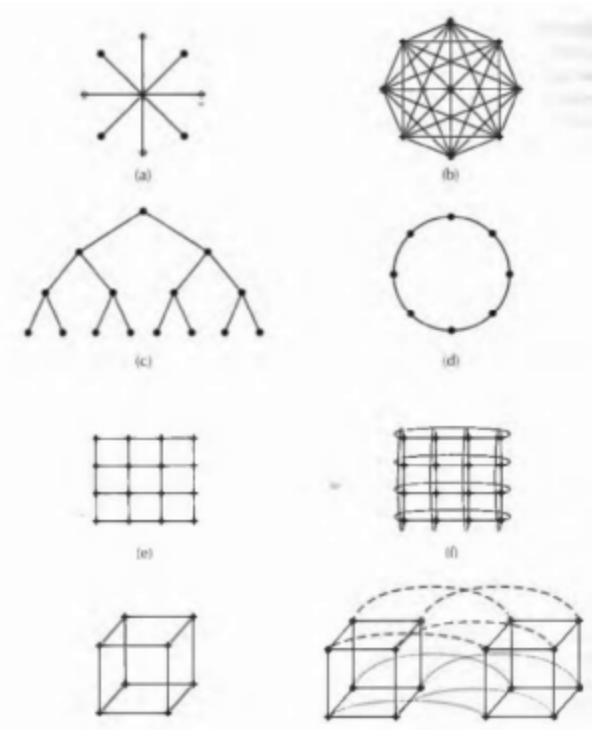
I messaggi sono spediti in un tempo nell'ordine dei μ sec, mille volte in meno rispetto ad un accesso in memoria (ordine dei ns). Il progetto è quindi più semplice da realizzare ed allo stesso tempo economico.

Hardware dei multicomputer

La topologia di una rete di interconnessione descrive il modo in cui sono disposti i collegamenti e i commutatori.

Il nodo base di un multicomputer consiste di una CPU, una memoria e una interfaccia di rete (talvolta anche un hard disk). Ogni nodo ha una interfaccia di rete con uno o due cavi (o fibra) che lo connette agli altri nodi oppure agli switch.

Alcune topologie possono essere:



Le memorie e le CPU non sono mostrate, ma si possono pensare collegate ai commutatori tramite interfacce.

In A troviamo la topologia a stella, in cui ci potrebbe essere uno switch attraverso il quale sono connessi tutti i nodi.

In D abbiamo un'altra rete di interconnessione, è la topologia ad anello: ogni nodo è connesso ad altri due nodi in ordine per formare un anello circolare (non sono necessari gli switch).

In E troviamo la griglia (grid) o maglia (mesh) è una struttura bidimensionale utilizzata in molti sistemi complessi. La sua forma regolare la rende altamente scalabile. Il percorso più lungo tra due nodi (diametro) aumenta come la radice quadrata del numero dei nodi.

In F troviamo il doppio toro, è una variante della griglia con i nodi estremi che si congiungono, è più tollerante ai guasti della maglia ma con un diametro più piccolo.

Schemi di switching

Nei multicomputer sono usati due tipi di schemi di switching:

Store-and-forward packet switching (connection-less)

Ogni messaggio è suddiviso in pacchetti che sono poi inseriti nella rete. Il pacchetto raggiunge il nodo destinatario attraverso delle politiche di instradamento che dipendono da vari fattori (traffico dati, priorità, ecc...).

Questo sistema è flessibile ed efficiente ma ha il problema dell'incremento dei tempi di latenza lungo la rete di interconnessione.

Circuit switching (connection-oriented)

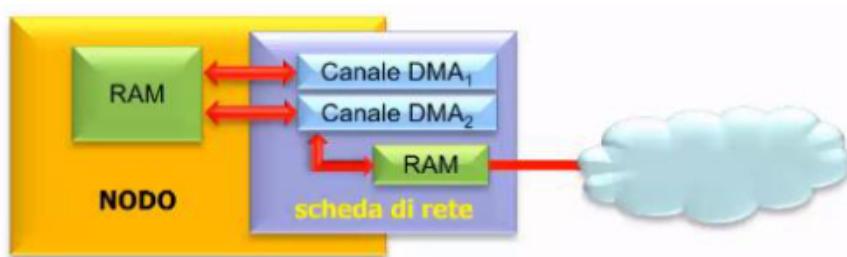
Nel secondo schema il primo switch stabilisce un collegamento fisico, attraverso tutti gli switch coinvolti, fino allo switch del nodo destinazione. Una volta che la connessione è creata i bit sono spediti alla massima velocità possibile (gli switch intermedi non hanno necessità di memorizzare i dati in transito).

Questo sistema richiede una fase di inizializzazione che prende tempo, ma una volta terminata il processo è velocissimo.

Una variante del circuit switching è il wormhole routing, spezza il pacchetto in sottopacchetti e permette a quest'ultimi di iniziare il tragitto prima che sia stato inizializzato il collegamento.

Interfacce di rete

In un multicomputer tutti i nodi hanno (almeno) una scheda di rete che permette di collegare il nodo alla rete di interconnessione, l'interfaccia contiene una RAM per memorizzare i pacchetti che entrano ed escono dal nodo.

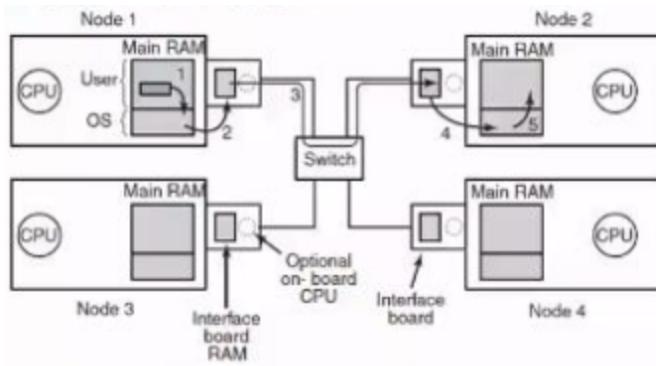


La scheda d'interfaccia può avere uno o più canali DMA oppure una CPU completa (processori di rete). I canali DMA possono copiare i pacchetti tra la scheda d'interfaccia e la RAM principale ad alta velocità, richiedendo i trasferimenti dei blocchi sul bus di sistema.

Software di comunicazione di basso livello

Il nemico delle prestazioni dei multicomputer è l'eccessiva copia di pacchetti.

Nell'esempio di sotto sono necessarie 5 copie prima di spedire il pacchetto dal nodo 1 al nodo 2.



Per evitare questo problema alcuni multicomputer mappano la scheda d'interfaccia nello spazio utente permettendo così al processo utente di inserire i dati direttamente nella scheda di rete (senza coinvolgere il kernel).

Questa soluzione pone due problematiche:

- La competizione dei processi concorrenti sullo stesso nodo che vogliono spedire pacchetti;
- La condivisione della scheda di rete tra il kernel, che magari vuole accedere ad un file system remoto, e il processo utente.

Riguardo la prima problematica, è meglio avere un solo processo per nodo.

Riguardo la seconda è meglio utilizzare due schede di rete per ciascuna funzione.

Software di comunicazione a livello utente

I processi sulle diverse CPU di un multicomputer comunicano attraverso lo scambio di messaggi. Il sistema operativo fornisce le primitive che consentono ai processi utente di inviare (send) e ricevere messaggi (receive).

- send (destination, &message_pointer): spedisce il messaggio indirizzato da message_pointer ad un processo identificato da destination;
- receive (address, &message_pointer): Il parametro address specifica l'indirizzo su cui il ricevente è in ascolto.

In un multicomputer statico il numero di CPU è noto a priori, quindi il campo address si compone di: Identificativo della CPU e Identificativo del processo o della porta sulla CPU selezionata.

Primitive bloccanti e non bloccanti

Le primitive send() e receive() possono essere bloccanti (sincrone) o non bloccanti (asincrone).

Se una send() è non bloccante, essa restituisce il controllo al chiamante immediatamente dopo la sua esecuzione e prima che il messaggio venga realmente spedito. Nel caso contrario il chiamante rimane in attesa finché non è inviato.

La scelta tra bloccanti o non dipende dal progettista del sistema sebbene in pochi sistemi siano entrambe disponibili.

Il vantaggio in termini di performance offerto dalle chiamate non bloccanti viene controbilanciato da un serio svantaggio: il processo che ha spedito il messaggio (attraverso la send()) non può accedere al buffer fino a che il buffer non è vuotato e il messaggio spedito. In più non conosce quando la trasmissione sarà terminata.

Esistono 3 possibili soluzioni:

1. Il kernel copia il messaggio in un buffer interno, quindi ogni messaggio è copiato dallo spazio utente a quello del kernel (eccesso di copie).
2. Al termine della spedizione il mittente riceve un interrupt in modo che possa riutilizzare li buffer (la gestione degli interrupt a livello utente è complessa e introduce ulteriori race condition).
3. Fare una copia del buffer nel caso in cui venga riscritto da una nuova spedizione modificano la sua pagina come read-only finché il messaggio non p completamente inviato (eccesso di copie).

Il processo mittente può eseguire quindi:

1. Una spedizione bloccante e mantenere bloccata la CPU;
2. Una spedizione non bloccante con copia (la CPU spreca tempo solo per eseguire una copia);
3. Una spedizione non bloccante con interrupt (programmazione difficile);

4. Una spedizione non bloccante con copia su scrittura (la CPU spreca tempo anche per la copia di fine processo oltre le scritture richieste).

In un sistema multi-thread la prima è la scelta migliore: mentre il thread che esegue la send è bloccato, gli altri continuano a lavorare.

Il processo destinatario può utilizzare una receive() non bloccante: indica semplicemente al kernel dove è il buffer.

L'arrivo di un messaggio può essere gestito:

- Tramite interrupt, ma sono difficili da programmare e molto lenti;
- Richiamando una procedura poll() che restituisce se ci sono messaggi in attesa di essere letti;
- Attraverso la creazione automatica di un thread (chiamato thread pop-up) che finito il suo compito muore spontaneamente;
- Attraverso un interrupt che attiva nella ISR il codice di gestione (questo schema è una versione ibrida dei precedenti che sfrutta l'idea di un thread pop-up senza creare alcun thread, migliorando così le performance, e si chiama messaggi attivi).

Remote Procedure Call (RPC)

Sebbene il modello a scambio di messaggi è un sistema conveniente per un sistema operativo multicomputer soffre di un grave difetto: tutte le comunicazioni, e quindi i programmi, utilizzano l'I/O.

Un approccio differente è quello che consente ai programmi di chiamare procedure che si trovano su le altre CPU del multicomputer in modo indipendente dall'I/O. Questa tecnica è detta RPC (Remote Procedure Call) ed è alla base del software del multicomputer.

Per chiamare una procedura remota il programma client deve avere una piccola procedura chiamata client sub. Analogamente, il programma server è legato ad una procedura chiamata server stub.

I passi effettivi per fare una RPC sono:

1. Il programma client chiama il client stub (locale);
2. Il client stub confeziona un messaggio e chiede al sistema operativo di spedirlo (marshaling).
3. Il kernel spedisce il messaggio dal nodo client a quello server.

4. Il kernel del nodo destinazione passa il messaggio al server stub.
5. Il server stub chiama la procedura invocata dal nodo sorgente.

Insidie nell'uso delle RPC

1. Non è possibile utilizzare i puntatori come parametri poiché una chiamata a procedura remota risiede su un altro nodo (quindi un differente spazio di indirizzi).
2. Quando si utilizzano array come parametri è necessario specificare le dimensioni, a differenza di ciò che accade con i linguaggi debolmente tipizzati in cui non è necessario farlo.
3. Non è sempre possibile dedurre il tipo di dato dai parametri quindi attivare procedure/metodi che siano polimorfe come ad esempio la printf(che può avere più parametri).
4. Non è possibile utilizzare le variabili globali, in quanto non esiste un contesto comune.

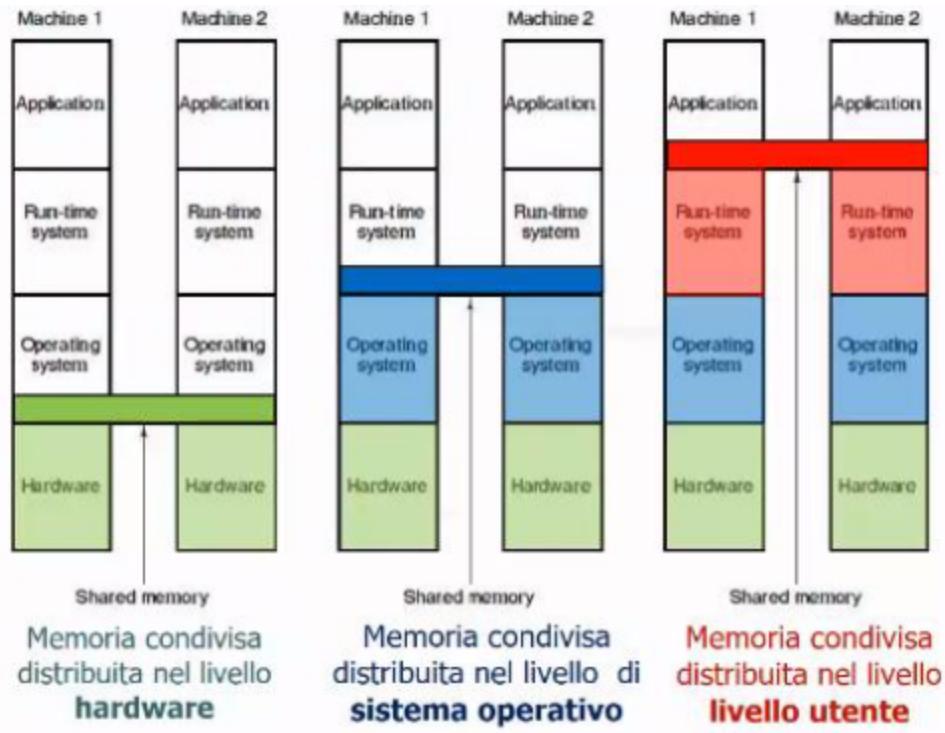
Memoria condivisa distribuita

Molti programmatore preferiscono un modello di memoria condivisa anche sui multicomputer. Attraverso la tecnica dei DSM (Distributed Shared Memory) è possibile fornire una versione astratta della memoria.

Ciascuna macchina ha la propria memoria virtuale. Quando una CPU legge o scrive su una pagina che non ha avviene una trap al sistema operativo:

- Prima il SO localizza la pagina.
- Chiede poi l'unmap della pagina alla CPU che la possiede e se la fa spedire nella rete di interconnessione
- Una volta ricevuta la pagina, viene rimappata e l'istruzione che ha generato l'errore riavviata.

La DMS si può collocare a diversi livelli dello stack:



Scheduling

Su un multiprocessore i processi risiedono nella stessa memoria: tutti i processi sono potenzialmente candidati all'esecuzione.

Su un multicomputer ogni nodo ha una propria memoria e un insieme di processi propri.

La schedulazione su un multicomputer è simile a quella su un multiprocessori, ma non tutti gli algoritmi dei primi si applicano ai secondi. L'algoritmo più semplice di tenere un unico elenco dei processi pronti, sui multiprocessori non funziona poiché ogni processo può essere eseguito solo sulla CPU corretta.

Quando si crea un nuovo processo occorre scegliere dove eseguirlo anche in relazione al bilanciamento del carico.

Bilanciamento del carico

Una volta che un processo assegnato ad un nodo, funziona qualsiasi algoritmo di scheduling locale (a meno che sia utilizzato il gang scheduling).

Diversamente dai multiprocessori, in cui tutti i processi condividono la stessa memoria e possono essere schedulati su qualsiasi CPU, nei multicomputer la scelta di quale processo spedire su un nodo è cruciale.

Gli algoritmo e le euristiche per l'assegnazione di un processo al nodo sono conosciuti come gli algoritmi di allocazione dei processi. Essi differiscono per i requisiti (CPU, memoria, traffico dati, ...) e per l'obiettivo.

7.g Virtualizzazione

La virtualizzazione è una tecnologia che ha più di 40 anni e consente a un singolo computer di ospitare più macchine virtuali. Ciascuna macchina virtuale ha un proprio sistema operativo.

Un sistema virtualizzato mantiene l'affidabilità di un sistema multicomputer ad un costo ridotto ed una maggiore semplificazione della manutenzione. Un guasto sul server che gestisce le macchine virtuali produce un danno catastrofico rispetto a quello di un nodo in un multicomputer. La probabilità di guasti di natura hardware è enormemente più bassa rispetto a quella di natura software.

La virtualizzazione introduce importanti vantaggi:

- Forte isolamento tra le macchine: un malfunzionamento su una macchina virtuale non inficia il comportamento nelle altre.
- La riduzione delle macchine fisiche riduce lo spazio, il consumo di energia, la produzione di calore quindi l'energia di raffreddamento.
- La creazione di checkpoint e la migrazione di macchine virtuali è più semplice rispetto ad un ambiente tradizionale
- Si possono far girare applicazioni legacy su ambienti obsoleti che non funzionerebbero con gli attuali hardware;
I programmati possono il test delle applicazioni su differenti SO senza disporre dell'hardware fisico necessario e SO.

Ogni CPU con una modalità kernel e una modalità utente ha un insieme di istruzioni.



Una macchina è virtualizzabile (tipo 1) solo se le tutte le istruzioni sensibili privilegiate.

A differenze del IBM/370 che è virtualizzabile (tipo 1), l'Intel 386 possiede delle istruzioni non privilegiate che sono ignorate se eseguite in modalità utente (come ad esempio la POPF). Quindi Intel 386 e i suoi successori fino al 2005 non possono essere virtualizzati con un hypervisor di tipo 1.

Esistono due differenti approcci per il monitor delle macchine virtuali:

Una macchina è virtualizzabile solo se le istruzioni sensibili sono un sottoinsieme di quelle privilegiate.

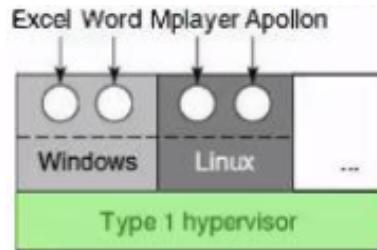
A differenza del IBM/370, l'Intel 386 possiede delle istruzioni sensibili che sono ignorate se eseguite in modalità utente (come ad esempio la POPF).

Quindi Intel 386 e i suoi successori non possono essere virtualizzati con un hypervisor di tipo 1.

Il problema viene risolto nel 2005, quando Intel e AMD introducono la virtualizzazione sulle loro CPU. Sulle CPU Intel la tecnologia è chiamata Virtualization Technology (VT) mentre sulle macchine AMD è denominata Secure Virtual Machine (SVM).

Sebbene entrambe si ispirano al funzionamento dell'IBM/370 hanno delle piccole differenze. L'idea fondamentale è creare dei contenitori all'interno dei quali eseguire le macchine virtuali. Quando in un computer si avvia un sistema operativo guest, questo continua a funzionare fino a che non solleva un'eccezione e passa un trap all'hypervisor.

Hypervisor di tipo 1



L'hypervisor di tipo 1 è nel SO reale (SO host) e gira in modalità kernel. Il suo compito è di supportare le macchine virtuali (SO guest) così come accade per i thread e i processi.

La macchina virtuale è eseguita come un processo utente in modalità utente, non può eseguire istruzioni sensibili anche se pensa di essere in modalità kernel (modalità virtuale del kernel).

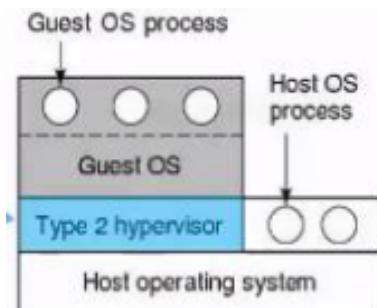
Quando il SO guest esegue una istruzione sensibile:

- Se la CPU non ha la VT la virtualizzazione non è possibile;
- Altrimenti avviene una trap nel kernel e l'hypervisor può vedere se l'istruzione è stata inviata da una VM del SO guest, in questo caso la esegue, o da un programma utente, in questo caso simula il comportamento dell'hardware reale.

Nel 2005 Intel e AMD introducono la virtualizzazione sulle loro CPU e rendono possibile il monitor delle macchine virtuali tipo 1. Sulle CPU Intel la tecnologia è chiamata Virtualization Technology (VT) mentre sulle macchine AMD è denominata Secure Virtual Machine (SVM).

Sebbene entrambe si ispirano al funzionamento dell'IBM/370 hanno delle piccole differenze: le macchine virtuali sono eseguite all'interno di contenitori finché il SO guest non provoca un'eccezione che genera un trap per l'hypervisor.

Hypervisor di tipo 2



Nell'hypervisor type 2 è un programma utente che interpreta le istruzioni della VM e le traduce sul SO della macchina reale: sono analizzati frammenti di codice insieme con lo scopo di incrementare le performance. È la soluzione che permette la virtualizzazione alle CPU Intel senza tecnologia VT (es. 386..).

VMware è un hypervisor di tipo 2 ed è eseguito come programma utente su un qualsiasi SO host.

Quando si avvia per la prima volta, si comporta come un computer senza SO e cerca di installare il SO guest nel suo disco virtuale.

Per eseguire un programma si utilizza una tecnica nota come traduzione binaria:

- Esegue la scansione del codice alla ricerca dei blocchi base, cioè quei blocchi che terminano con istruzioni di cambio flusso (es. JMP, CALL, trap,...);
- Ricerca nei blocchi le istruzioni sensibili e le traduce in una procedura VMware;
- Il blocco è messo nella cache di VMware e può essere eseguito alla velocità della macchina fisica (istruzioni tradotte a parte).

Confronto tra hypervisor

Negli hypervisor tipo 2 tutte le istruzioni sensibili sono sostituite da chiamate a procedure che ne emulano il comportamento. Il SO guest non invierà mai nessuna istruzione sensibile alla macchina fisica.

L'approccio “trap-and-emulate”(tipo 1) adottato dagli hardware VT genera troppi trap ed un eccessivo overhead di gestione, mentre la traduzione delle istruzioni sensibili è più efficiente.

Alcuni hypervisor di tipo 1 effettuano la traduzione binaria (anche se non serve) comportandosi come quelli di tipo 2.

Paravirtualizzazione

Gli hypervisor tipo 1 e 2 funzionano senza modifiche al SO guest, ma con performance non eccellenti.

Un diverso approccio prevede la modifica del codice sorgente del SO guest: invece di eseguire istruzioni sensibili si effettuano chiamate di procedure definite dall'hypervisor.

Quindi l'hypervisor definisce una interfaccia, cioè delle API (Application Program Interface), che i sistemi operativi guest possono attivare.

Questo trasforma di fatto l'hypervisor in un microkernel e il SO guest modificato viene detto paravirtualizzato. Le performance ovviamente migliorano poiché le trap si trasformano in system call.