

Architettura degli elaboratori

Sommario

Sommario	1
Differenza tra traduzione e interpretazione	3
CPU	6
CISC e RISC	7
Dischi Magnetici	8
Hard disk	8
Hard disk magnetici	8
Dischi IDE	8
SCSI disks	9
Dischi a stato solido (SSD)	9
CD-ROM	9
Raid	9
Negazione NOT	12
Congiunzione logica AND	12
Disgiunzione logica OR	13
Disgiunzione logica esclusiva XOR o EXOR	13
Circuiti logici	15
Multiplexer	16
Decoder(decodificatori)	16
PLA(Programmable Logic Arrays)	17
Circuiti aritmetici - Lo shifter	17
Half Adder (Semi-sommatore)	17
Full Adder (Sommatore)	18
Clock	18
Latch	18
SR Latch clocked	18
Flip-Flop	18
Bus	19

Bus sincroni	19
Bus asincrono	19
Arbitraggio del bus	19
Dispositivi Input/Output	20
Livello di Microarchitettura	20
Percorso Dati (Data Path)	20
ALU (Arithmetic Logic Unit)	21
Temporizzazione del percorso dati	22
Incremento di un registro (Un ciclo di clock)	22
Microistruzioni	22
Cos'è una microistruzione?	23
Il formato delle microistruzioni	23
Micro Architettura Mic-1	24
Il funzionamento di Mic-1	25
Il livello ISA e un esempio di ISA: IJVM	25
Lo stack	25
Il modello della memoria IJVM	26
Insieme delle istruzioni della IJVM	27
I Registri	27
Intel Core i7	28
Overview del livello ISA del Core i7	28
Modalità operative di Intel Core i7	28
Registri di Intel Core i7	28
Architetture per il calcolo parallelo	30
Parallelismo	31
Parallelismo a livello delle istruzioni:	31
Multithreading nel chip:	31
Multiprocessori in un solo chip:	32
Parallelismo a livello delle istruzioni	32
Pipelining	32
Il problema dello stallo della pipeline	33
Multithreading	34
Multithreading a grana fine	34
Multithreading a grana grossa	35
Multithreading a CPU superscalari	35
Multithreading simultaneo	36
Multiprocessori in un solo chip	36
Multiprocessori omogenei in un solo chip	36
Multiprocessori eterogenei in un solo chip	37
Chip-level Multiprocessor	37
Coprocessori	37
MIMD	37
Multiprocessori e Multicomputer (MIMD)	38
Definizione Multiprocessore:	38
Definizione Multicomputer:	38

Hardware dei multiprocessori (UMA e NUMA)	38
UMA con architettura basata su bus	39
UMA con singolo bus e cache nelle CPU	39
UMA con singolo bus e CPU dotate di RAM	39
UMA con crossbar switch	40
UMA con rete di commutazione a più stadi (o livelli)	41
Multiprocessori NUMA	41
Tipi di SO multiprocessore	42
Ogni CPU ha il SO personale	42
Multiprocessore Master-Slave	42
Multiprocessori Simmetrici	42
Sincronizzazione dei multiprocessori	43
Scheduling dei multiprocessori	43
Timesharing	43
Condivisione dello spazio(Space Sharing)	44
Schedulazione Gang	44
Multicomputer	44
Hardware dei multicomputer (Topologia)	45
Store-and-forward packet switching	47
Circuit switching	47
Interfacce di rete	47
Software di comunicazione a basso livello	48
Software di comunicazione a livello utente	48
Send e Receive	48
Primitive bloccanti e non bloccanti	48
Remote Procedure Call(Chiamate di procedura remote)	50
Problemi Implementativi	50
DSM(Distributed Shared Memory)	50
Virtualizzazione	51
vantaggi della virtualizzazione	51
Tipi di virtualizzazione	51
Confronto tra hypervisor	53
Paravirtualizzazione	53

Differenza tra traduzione e interpretazione

Un metodo per eseguire un programma scritto in L1 consiste nel sostituire, in una fase iniziale, ogni sua istruzione con un'equivalente sequenza di istruzioni in L0. Il programma che ne risulta è costituito interamente da istruzioni di L0 e può essere eseguito dal computer al posto del programma L1 originale. 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, che non richiede la generazione preventiva di un nuovo programma L0, è chiamata **interpretazione** e il programma che la esegue è detto interprete.

la traduzione: è tradurre tutto il programma al linguaggio inferiore e poi viene eseguito.

l'interpretazione: traduce ed esegue ogni singola istruzione del programma. Legge ed esegue il codice sorgente del programma senza creare un file oggetto eseguibile. E' più lenta rispetto alla compilazione.

Differenze tra compilazione e interpretazione

la compilazione e l'interpretazione sono due metodi per eseguire un programma informatico.

La compilazione traduce tutte le istruzioni di un programma in linguaggio macchina, creando un file eseguibile dal computer.

La compilazione viene eseguita da un software compilatore.

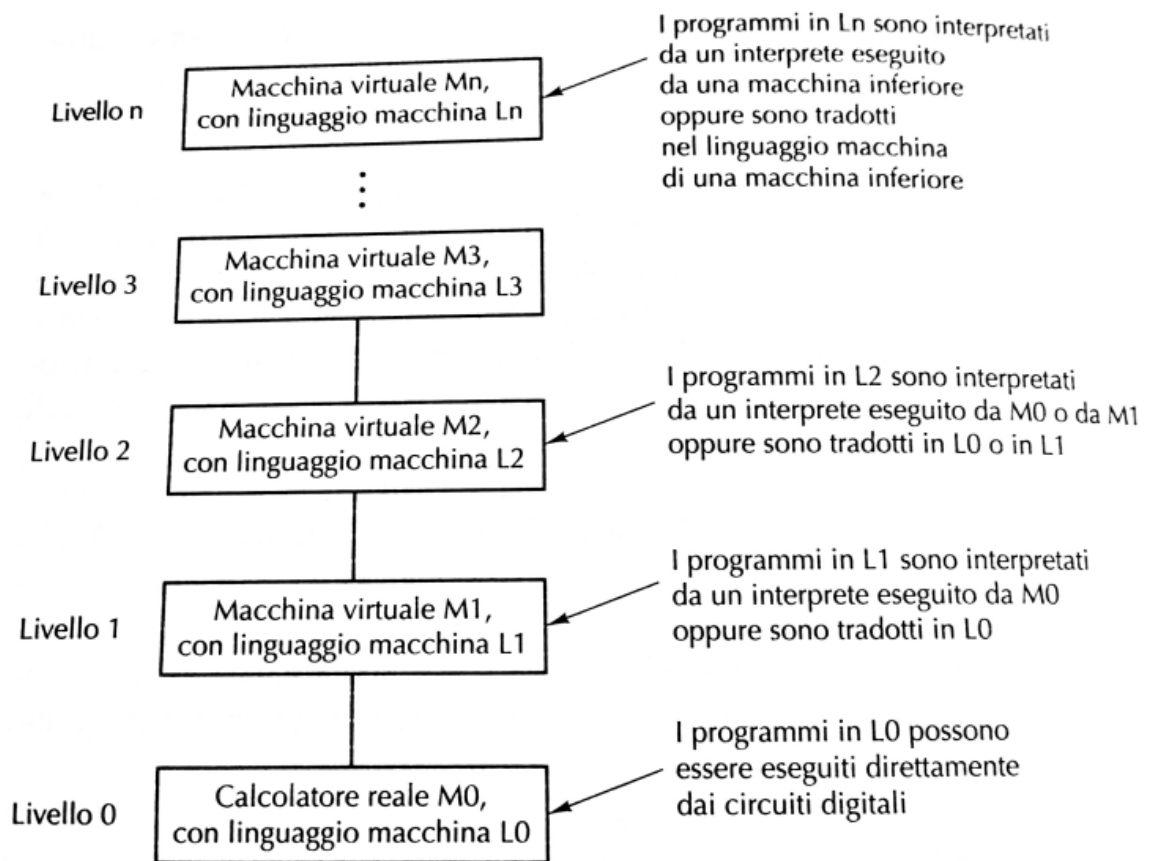
L'interpretazione traduce ed esegue ogni singola istruzione del programma. Legge ed esegue il codice sorgente del programma senza creare un file oggetto eseguibile. E' più lenta rispetto alla compilazione.

DIFFERENZE

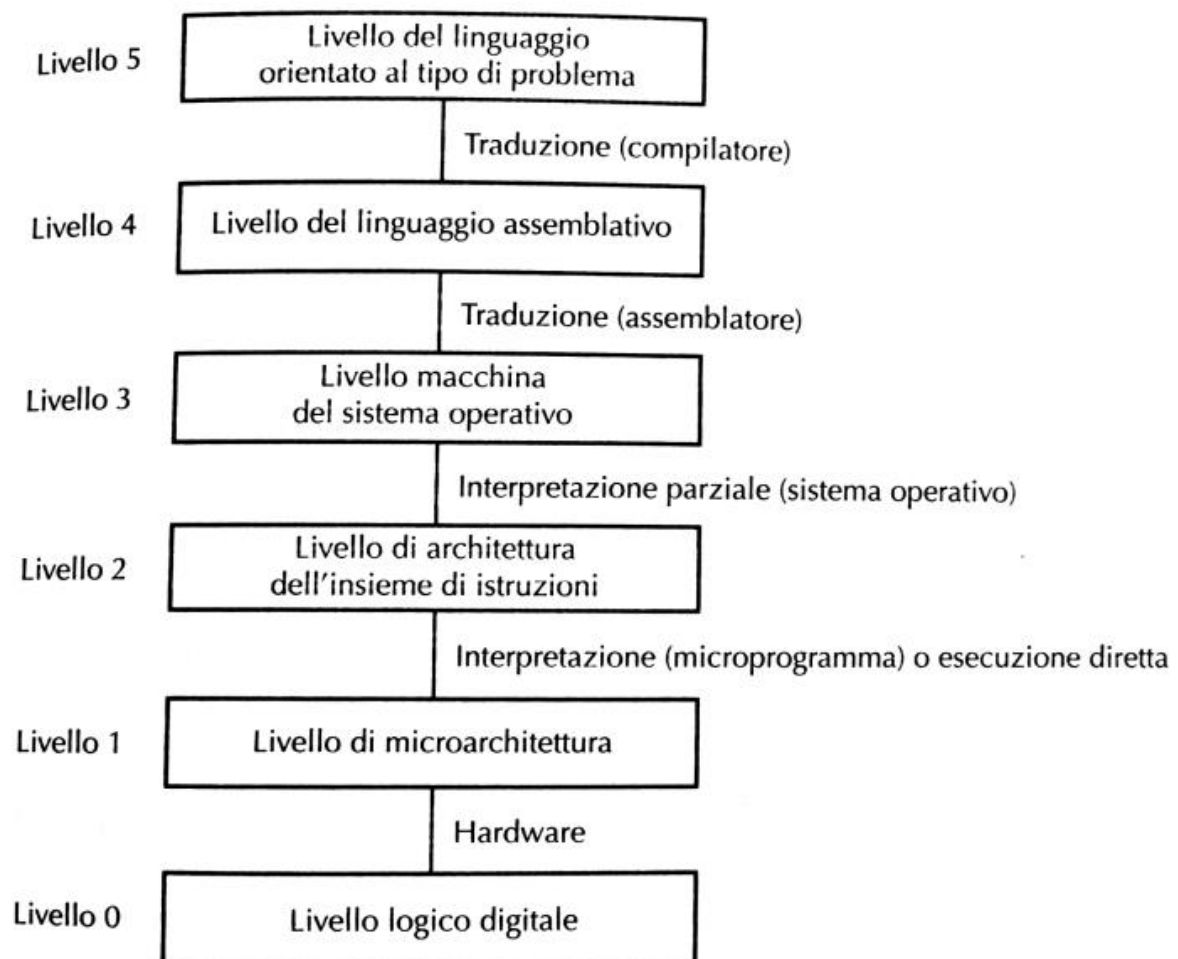
L'esecuzione di un programma compilato è molto più veloce rispetto a un programma interpretato. Il motivo è molto semplice, *la compilazione crea un file eseguibile (oggetto) in linguaggio macchina ed è immediatamente eseguibile dal processore.*

Viceversa, *l'interpretazione legge un'istruzione alla volta dal codice sorgente del programma (sorgente) e la traduce in linguaggio macchina per farla eseguire dal processore.*

Dopo aver eseguito un'istruzione, il software interprete passa ad elaborare la successiva e non resta traccia della precedente. Tutti questi passaggi si ripetono ogni volta che eseguo il programma. Per questa ragione l'esecuzione di un programma interpretato è più lenta.



livelli di un computer



CPU

Una tipica CPU di Von Neumann è formata da dei registri (1 a 32), dalla ALU e da alcuni bus che connettono tra loro diverse parti. I registri alimentano due registri di input della ALU che mantengono i dati d'ingresso della ALU. La ALU esegue operazioni, somma e sottrazione, e il risultato viene memorizzato in un apposito registro di input, successivamente può essere immagazzinato in uno dei registri della CPU e volendo copiato in memoria.

Le operazioni sono divise in: istruzioni registro-memoria e istruzioni registro-registro.

Le prime prelevano parole in memoria e vengono portate all'interno dei registri, utilizzabili per la ALU per operazioni successive.

Le seconde preleva due operandi dai registri e li porta dentro dei registri di input della ALU e ne memorizza il risultato in uno dei registri. Il processo che porta i due operandi attraverso la ALU e memorizza il risultato è chiamato ciclo del percorso dati ed è il cuore della CPU.

La CPU esegue ogni istruzione facendo:

1. preleva la successiva istruzione dalla memoria per portarla nell'IR;

2. modifica il PC per farlo puntare all'istruzione seguente;
3. determina il tipo di istruzione;
4. se l'istruzione usa una parola in memoria, determina dove si trova;
5. se necessario, preleva la parola per portarla in un registro della CPU;
6. esegue l'istruzione;
7. torna al punto 1 per l'istruzione successiva.

Questa serie di passi viene detta ciclo esecutivo delle istruzioni.

CISC e RISC

La **CISC** è formata da un set di istruzioni in grado di eseguire operazioni molto più complesse rispetto all'architettura RISC. E' infatti possibile utilizzare una sola istruzione per leggere un dato in memoria, modificarlo e salvarlo nuovamente, cosa impensabile con un set di istruzioni ridotto. Sebbene questo tipo di architettura non abbia la velocità di quella RISC, ha il vantaggio di poter realizzare programmi più compatti che occupano un minor spazio in memoria. I programmi in questione sono però più "pesanti", e richiedono molto tempo per l'esecuzione. Un singolo set di istruzioni è eseguito in più passaggi; ogni set di istruzioni ha oltre 300 istruzioni separate. Le istruzioni vengono completate in 2-10 cicli macchina.

La **RISC** è semplice e lineare, permettendo al microprocessore di eseguire il set di istruzioni in tempi molto rapidi e inferiori ai tempi dell'architettura CISC. Esiste un insieme di principi di progettazione, chiamati principi di progettazione RISC, che i progettisti delle CPU cercano di seguire il più possibile

- Tutte le istruzioni sono eseguite direttamente dall'hardware;
- Massimizzare la frequenza di emissione delle istruzioni;
- Le istruzioni devono essere facili da decodificare;
- Solo le istruzioni Load e Store fanno riferimento alla memoria;
- Molti registri disponibili.

CISC vs RISC

CISC	RISC
Enfasi sull'hardware	Enfasi sul software
Istruzioni di diverse dimensioni e formati	Istruzioni della stessa dimensione con pochi formati
Meno registri	Usa più registri
Più modalità indirizzamento	Meno modalità indirizzamento
Uso esteso della microprogrammazione	Complessità nel compilatore
Le istruzioni richiedono un numero variabile di cicli di clock	Le istruzioni richiedono un solo ciclo di clock
Il pipelining è difficoltoso	Il pipelining è semplice

Dischi Magnetici

Ogni traccia è divisa in settori di lunghezza fissata, tra settori c'è un piccolo spazio chiamato intersector gap. Le tracce interne hanno maggiore densità di memorizzazione rispetto alle esterne. Ogni drive ha una scheda dedicata (può contenere una CPU) chiamato controllore del disco, oltre a pilotare accetta i comandi software, corregge i dati.

Hard disk

Sono dispositivi di memorizzazione dati che possono essere collegati internamente o esternamente al calcolatore.

Hard disk magnetici

Costituito da una pila di dischi che ruotano intorno all'asse, l'insieme di tracce è detto cilindro.

Le performance dell'HD dipendono da:

- tempo medio di seek;
- latenza rotazionale;
- tempo di trasferimento.

Il posizionamento della testina nel raggio della traccia ricercata è detto tempo medio di seek. La latenza rotazionale è il tempo necessario al disco per posizionare il corretto settore sotto la testina e dipende dalla velocità angolare del disco.

Il tempo di trasferimento dipende dalla densità lineare e dalla velocità di rotazione.

Dischi IDE

Il controller dell'Hard disk è su una scheda separata, fu utilizzato lo standard IDE in grado di gestire dischi con una capacità fino a 504 MB.

Il SO leggeva e scriveva dati sul disco inserendo parametri nei registri della CPU e poi invocando il BIOS. Da IDE si passò all'EIDE dove ha uno schema di indirizzamento aggiuntivo denominato LBA. Ai controllori EIDE potevano avere due canali su ciascuno dei quali poteva essere collegato un driver primario e secondario.

Dopo EIDE ci fu ATA-3, poi ATAPI-4 incrementarono la dimensione del connettore e la velocità di trasferimento, ATAPI-5 e ATAPI-6, mentre ATAPI-7 incrementa la dimensione del connettore per aumentare la banda, utilizza un serial ATA per trasferire i bit su un connettore.

SCSI disks

I controller possono collegare fino a 7 dispositivi, hanno una organizzazione del tutto simile ai dischi IDE ma necessitano di differenti interfacce e hanno più alte velocità

Dischi a stato solido (SSD)

Sono dischi basati su memoria flash non volatile, hanno prestazioni eccellenti, la velocità è fino a tre volte superiore rispetto ad un HD magnetico.

CD-ROM

è fatto di polycarbonato poi una striscia sottile di alluminio riflettente. Nel substrato di polycarbonato è possibile creare delle depressioni tra aree non incise.

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 si usa un laser a bassa potenza.

Per memorizzare informazioni esistono due modalità: con correzione errori o senza ECC.

Un settore contiene:

- un preambolo
- la sezione dei dati
- il codice ECC

il modo senza ECC è utilizzato per applicazioni audio/video.

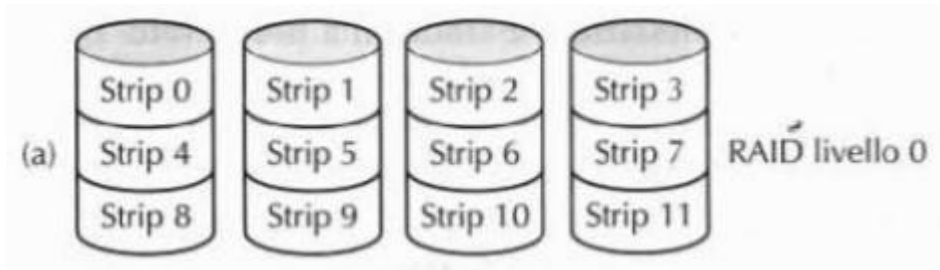
Raid

Per migliorare le prestazioni dei dischi (anche l'affidabilità) si pensò di adottare la strategia del calcolo parallelo.

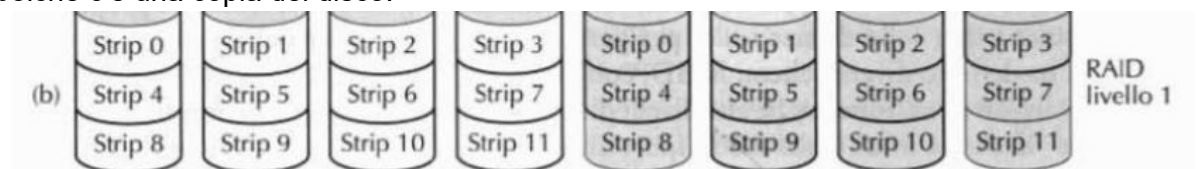
L'idea è di far vedere al calcolatore il sistema RAID (insieme di dischi) come un unico enorme disco virtuale con elevata performance e affidabilità.

Esistono 5 schemi RAID:

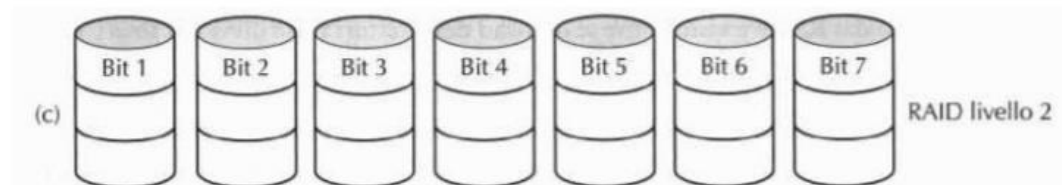
- **RAID Livello 0:** i dati sono suddivisi in strisce di k settori e memorizzati in dischi con modalità round-robin. In questo modo un blocco di dati può essere letto con 4 letture parallele. Lavora meglio con richieste di grandi dimensioni, lavora male se il SO richiede i dati solo ad un settore alla volta;



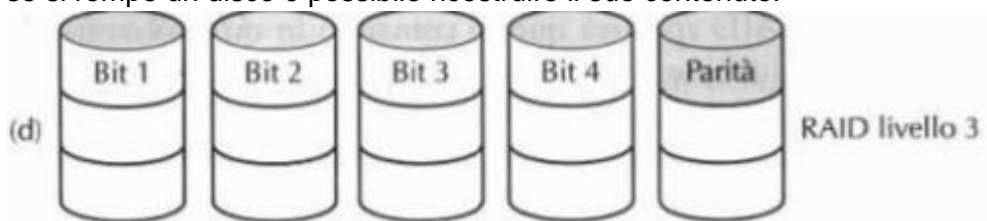
- **RAID Livello 1:** Stesso funzionamento RAID 0 ma dischi duplicati, così ci sono tanti dischi primari quanti di backup. In scrittura ogni striscia è scritta due volte, in lettura possono essere utilizzate tutte le copie, distribuendo il carico su più dischi. Se un disco si rompe, si può utilizzare una replica, il ripristino di un disco rotto è semplice poiché c'è una copia del disco.



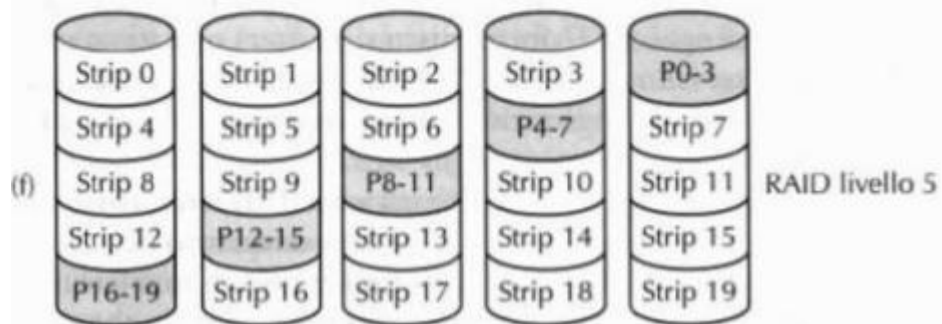
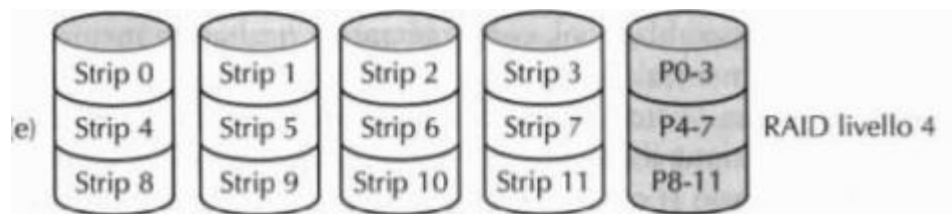
- **RAID Livello 2:** Utilizza parole binarie, o byte, per decomporre le informazioni sui dischi. Con 7 dischi è possibile dividere un bit per disco: i byte si dividono in 4 bit e si aggiungono 3 bit ottenendo così un codice a correzione di errore di hamming. Se si rompe un disco si può ricostruire facilmente. Il throughput è alto poiché le operazioni sono in parallelo. La rotazione dei dischi sia sincronizzata e che si utilizzino un numero elevato di dischi. Carico di lavoro del controller per calcolare hamming è elevato;



- **RAID Livello 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 Livello 4/5:** I RAID 4 e 5 lavorano su strisce e non richiedono dischi sincronizzati. È come il RAID 0, con una parità striscia per striscia scritta su un disco separato di parità. Si esegue l'EXOR bit a bit di tutte le strisce ottenendo così una striscia di parità.



Negazione NOT

“Il risultato è 1 se l’operando in ingresso è 0, mentre il risultato è 0 se l’operando è 1”

A	\bar{A}
0	1
1	0

- Eliminazione di un numero pari di negazioni $\bar{\bar{A}} = A$

Congiunzione logica AND

“Il risultato è 1 se e solo se tutti gli operandi sono 1”, rispetto al valore 0 “il risultato è 0 se e solo se esiste almeno un operando al valore 0”

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

(*** != negato***)

- Idempotenza $A * A = A$
- Commutativa $A * B = B * A$
- Associativa $A(B * C) = (A * B)C$
- Elemento neutro $1 * A = A$
- Elemento assorbente $0 * A = 0$
- Complementi $A! * A = 0$

Disgiunzione logica OR

“Il risultato è 1 se e solo se almeno un operando è 1”, rispetto al valore 0 “il risultato è 0 se e solo se tutti gli operandi sono 0”

A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1

- Idempotenza $A + A = A$
- Commutativa $A + B = B + A$
- Associativa $A+(B+C) = (A+B)+C$
- Elemento neutro $0 + A = A$
- Elemento assorbente $1 + A = 1$
- Complementi $A! + A = 1$

Disgiunzione logica esclusiva XOR o EXOR

“Il risultato è 1 se e solo se un numero dispari di operandi valgono 1”, rispetto a 0 “il risultato è 0 se e solo se un numero pari di operandi valgono 1”

A	B	$A \dot{\vee} B$
0	0	0
0	1	1
1	0	1
1	1	0

(*** != NEGAZIONE***)

- Conversione $A \oplus B = \bar{A} * B + A * \bar{B}$

è semplificabile? NO. infatti facendo la mappa di karnaugh si può osservare che non è semplificabile dato che ogni 0 è seguito da un 1 (a destra o in base alla sua posizione) e ogni 1 è seguito da uno 0 (a destra o in base alla sua posizione)

Altre proprietà:

- Distributiva di AND rispetto ad OR:

$$A * (B + C) = (A * B) + (A * C)$$

- Distributiva di OR rispetto ad AND:

$$A + B * C = (A + B) * (A + C)$$

- Primo teorema dell'assorbimento:

$$A * (A + B) = A$$

- Secondo teorema dell'assorbimento:


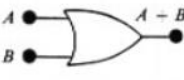
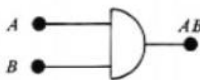
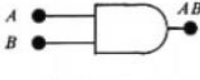
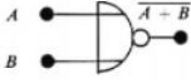

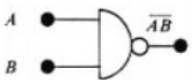
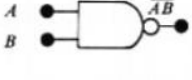

$$A * (\bar{A} + B) = A * B$$

- Legge o teorema di De Morgan:

$$\overline{A * B} = \bar{A} + \bar{B}$$

Circuiti logici

TABELLA VIII. – ALCUNE FUNZIONI BINARIE DI DUE VARIABILI, A E B .

Somma logica o disgiunzione o OR	Prodotto logico o congiunzione o AND	NOR (NOT-OR)	NAND (NOT-AND)	OR esclusivo
$F = A + B$ $F = A \vee B$	$F = A \cdot B$ $F = A \wedge B$	$F = A \downarrow B =$ $= \overline{A + B}$ $= \overline{A} \cdot \overline{B}$	$F = A \uparrow B =$ $= \overline{A \cdot B} =$ $= \overline{A} + \overline{B}$	$F = A \oplus B =$ $= \overline{A}B + A\overline{B}$
 	 	 	 	

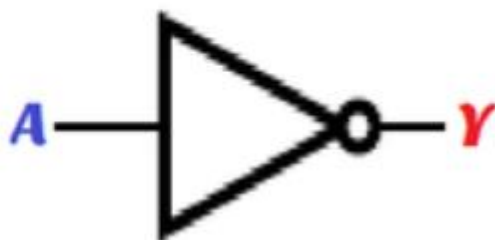
NOT



A	Y
0	1
1	0

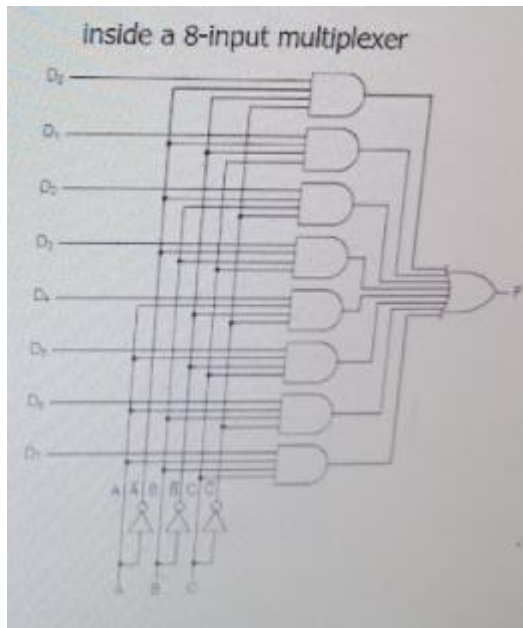
Porta logica NOT $Y = \overline{A}$

A	Y
0	1
1	0



Multiplexer

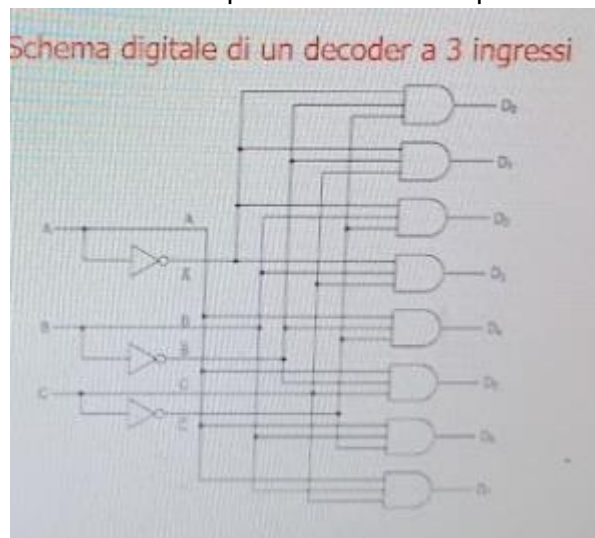
Il multiplexer è un circuito con 2^n ingressi, un' uscita e n ingressi di controllo che selezionano la linea in ingresso che verrà trasferita in uscita



a partire dalla forma normale disgiuntiva, si osserva che è possibile costruire un qualsiasi circuito utilizzando esclusivamente porte and,or,not, di conseguenza anche il mux può essere utilizzata per realizzare qualsiasi funzione logica

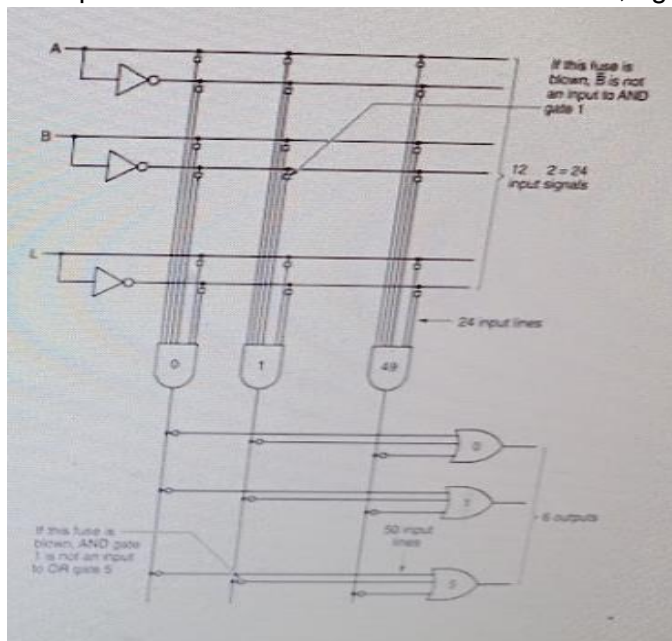
Decoder(decodificatori)

Circuito che riceve un numero a n-bit come ingresso e seleziona in uscita l'unica linea corrispondente al suo valore numerico. Utile quando si dispone di un indirizzo e si vuole selezionare il chip di memoria corrispondente.



PLA(Programmable Logic Arrays)

Un chip generico che permette di calcolare somme di prodotti. può essere costruito formando gruppi di AND e OR è la **PLA**. Esempio di PLA con 12 ingressi, il cuore è un array di 50 porte AND che creano una matrice 24x50, ogni linea di ingresso contiene un fusibile.



a partire dalla forma normale disgiuntiva, si osserva che è possibile costruire un qualsiasi circuito utilizzando esclusivamente porte and,or,not, di conseguenza anche la PLA può essere utilizzata per implementare una qualsiasi funzione.

Circuiti aritmetici - Lo shifter

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=sinistra, 1=destra)

Half Adder (Semi-sommatore)

Un sommatore è un circuito in grado di eseguire la somma tra bit, tra due variabili il circuito genera il risultato della somma e l'eventuale riporto. Viene chiamato semi-sommatore poiché non gestisce un riporto in ingresso.

Full Adder (Sommatore)

utilizzando due semi-sommatori si può costruire un sommatore che prende in ingresso tre bit (A,B e riporto in ingresso) e restituisce in ingresso somma e riporto.

Clock

è un circuito che emette una serie di impulsi di larghezza predefinita a intervalli di tempo costanti. L'intervallo di tempo compreso tra due fronti in salita è detto **ciclo di clock**. Per ottenere un'accuratezza elevata la frequenza di clock è controllata da un oscillatore a cristalli.

Latch

Per creare una memoria a 1 bit è necessario disporre di un circuito che in qualche modo ricordi i precedenti valori di input.

Un circuito semplice può essere realizzato con due porte logiche NOR(dipendono dai valori dei precedenti ingressi). Latch Set-Reset ha due ingressi: S(set) è utilizzato per impostare ad 1 il valore dell'uscita e R(Reset) usato per azzerarlo.

SR Latch clocked

è un latch in cui i segnali S e R agiscono solo quando il segnale di clock è attivo. è il primo circuito di memoria che ha la capacità di ricordare in Q e l'ultimo valore impostato per S o R

Flip-Flop

sono dei circuiti che permettono la creazione di registri di memoria se combinati fra loro.

Esempio 8 flip-flop creano un registro da 8 bit.

è sequenziale

Nei Latch la transazione avviene quando il segnale di clock è alto.

La transizione di stato non si verifica quando il clock vale 1 ma durante la transizione da 0 a 1 o da 1 a 0. Il flip flop è a commutazione su fronte mentre il Latch è a commutazione a livello .

La lunghezza dell'impulso del clock non ha alcuna importanza, purché le transizioni si verifichino con sufficiente velocità.

Bus

Un bus è un collegamento che unisce vari dispositivi, possono essere interni alla CPU (connessione registri ALU) o esterni (memoria o periferiche di I/O).

Oggi esistono più bus per il collegamento CPU-Memoria e CPU-device di I/O.

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

Per amplificare il segnale dei master sul bus è utilizzato un chip detto bus driver, viceversa gli slave usufruiscono di un chip chiamato bus receiver, per evitare di connettere più uscite insieme si utilizzano dei buffer tri-state oppure il collegamento in wired-OR di più porte open collector.

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 permette di indirizzare maggiori spazi di indirizzamento ma incrementa anche i costi, fili e connettori.

Per incrementare la larghezza di banda dei bus si può:

- ridurre il periodo di clock del bus, ma si corre il rischio di:
 - avere dispositivi che operano a velocità differenti;
 - perdere la retrocompatibilità.
- incrementare l'ampiezza del bus dati (più bit nell'unità tempo), per non avere bus troppo ampi si utilizza la tecnica del multiplexing ma porta ad un rallentamento del sistema.

Bus sincroni

utilizzano un clock che determina la temporizzazione delle attività sul bus, ogni operazione richiede un numero di periodi di clock per essere eseguita.

1. La CPU master pone l'indirizzo di memoria sull'address bus in modo che le linee si stabilizzano;
2. La CPU comunica al sistema che l'operazione che intende fare è con la memoria;
3. La CPU comunica che si tratta di un'operazione in lettura, così la memoria slave deve fornire il contenuto della cella indirizzata dall'address bus;
4. Poiché la memoria è meno veloce della CPU, c'è maggiore stato di attesa.

Bus asincrono

Il bus si adatta alla velocità del dispositivo collegato ed un dispositivo lento non rallenterà il sistema. La maggior parte dei bus sono sincroni perché più semplice da realizzare.

Arbitraggio del bus

Ciascun dispositivo intelligente del computer (CPU, coprocessori, ecc...) può diventare a turno master del bus. L'arbitraggio dei bus è utilizzato per prevenire situazioni di conflitto in cui due o più dispositivi tentano di diventare master.

Due tipi **centralizzato** o **decentralizzato**.

Il primo necessita di un arbitro, quando riceve una richiesta, concede la richiesta asserendo una linea di concessione del bus.

Quando il dispositivo più vicino vede la connessione:

- se lo ha chiesto lui, blocca la linea negandola a tutti;
- sennò mantiene asserita la linea.

Quando due o più fanno richiesta la ottiene il più vicino all'arbitro.

Per superare il problema che la priorità è cablata nella connessione molti bus definiscono più livelli di priorità utilizzando differenti linee richiesta-concessione, così l'arbitro concede il bus al dispositivo con priorità maggiore, a parità di priorità vince chi è più vicino a lui.

Il secondo, ogni dispositivo ha una propria linea di richiesta ed una priorità, prima di inviare una richiesta ciascuno deve verificare che non ci sia già una richiesta con priorità più alta. Al termine dell'utilizzo del bus, la linea deve essere negata. Lo svantaggio è che ci sono troppi collegamenti, il numero di collegamenti non può superare il numero di linee.

Lo schema utilizza tre linee:

- La linea di richiesta del bus;
- La linea che busy asserita dal dispositivo bus master corrente;
- La linea di arbitraggio propagata tra i dispositivi in cascata.

Per ottenere un bus un dispositivo deve:

- controllare che busy sia negata e che l'ingresso IN sia asserito;
- così nega OUT, asserisce la linea busy e diventa il master del bus.

Al termine sblocca Out e libera busy negandolo.

Dispositivi Input/Output

I dispositivi esterni dovrebbero rappresentare/raccogliere informazioni in una forma comprensibile agli essere umani.

Le periferiche esterne possono essere divise in tre categorie: input, output e input e output.

Quelli input permettono di inserire dati all'interno del computer (tastiera, mouse);

Quelli output permettono di fornire dati dal computer (monitor, stampante);

Quelli Input/Output permettono di inserire ed estrarre dati dal computer (modem, hard disk)

Livello di Microarchitettura

Percorso Dati (Data Path)

Il percorso dati è quella parte della CPU che contiene la ALU, i suoi input e i suoi output. Il data path contiene registri a 32 bit che controllano l'accesso in memoria. Inoltre il percorso dati si compone di due bus: B e C (l'operando A è quello contenuto nel registro H).

La maggior parte dei registri può inviare il proprio contenuto sul bus B, collegato in input alla ALU. Alla base dell'ALU troviamo lo shifter, il quale invia il proprio risultato al bus C.

I registri del data path sono i seguenti (**quelli in grassetto sono i più importanti**):

- **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 Variabile (LV), è il riferimento nello stack alla base delle variabili locali.

Gli altri registri presenti sono:

- Constant Pool (CPP)
- Top word On the Stack (TOS)
- Op Code register (OPC) è il registro temporaneo, può contenere l'ultima istruzione eseguita prima di un salto e identifica il tipo di istruzione indicando anche se è di tipo ADD o di tipo BRANCH
- Holding (H).

ALU (Arithmetic Logic Unit)

La ALU ha 6 linee di controllo:

- F0 e F1: selezionano il tipo di funzione
- ENA e ENB: abilita il valore della variabile A o B altrimenti lo annulla.
- INVA: esegue una sottrazione della variabile A.
- INC: incrementa.

La ALU agisce su due dati in ingresso: A (che viene dal registro H) e B.

E' 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.

La ALU contiene 3 unità differenti: un decoder, una unità logica e un full adder.

- Il decoder permette di selezionare l'operazione richiesta in base ai segnali F0 e F1.
- L'unità logica è in grado di calcolare: $A*B$, $A+B$ e la negazione di B.
- Il full adder somma A, B e il riporto in ingresso e calcola il risultato e l'eventuale riporto.

Le ALU ad 1 bit possono essere assemblate insieme per costruire un ALU di lunghezza variabile.

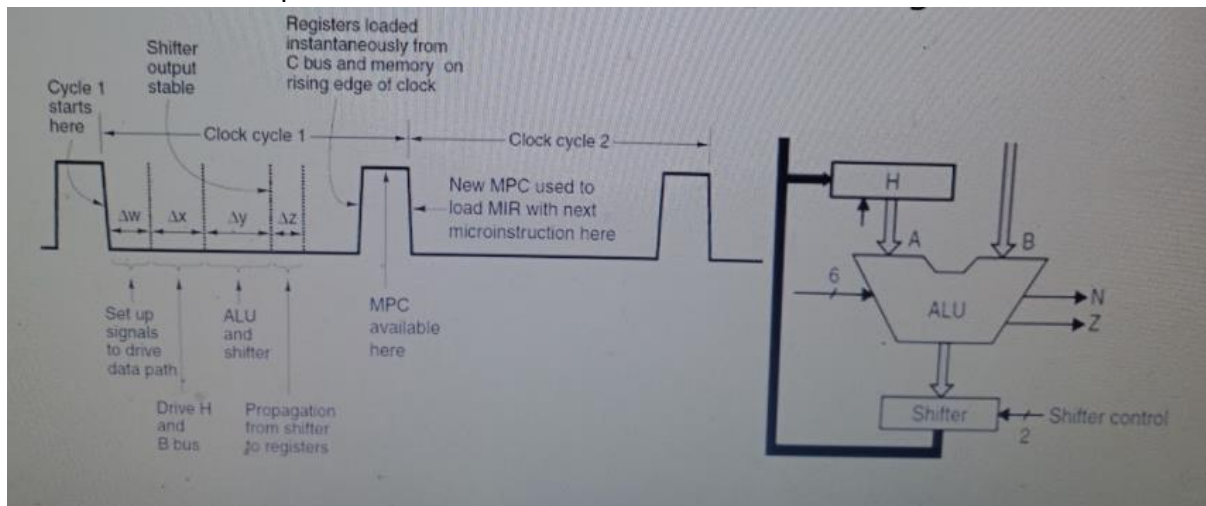
Questa tecnica è detta bit slice (suddivisione di bit) e può essere applicata anche agli altri circuiti digitali che lavorano bit a bit.

Temporizzazione del percorso dati

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.



Incremento di un registro (Un ciclo di clock)

- 1) Si pone il valore del registro sul bus B.
- 2) Si disabilita l'operando A e si incrementa B.
- 3) Non si effettua alcun scorrimento.
- 4) 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.

Cos'è una microistruzione?

Una microistruzione è una sequenza di bit che costituisce un microprogramma (microprogramma = sequenza di cicli del data path necessaria all'esecuzione dell'istruzione).

Il formato delle microistruzioni

il formato delle microistruzioni può essere diviso in 6 gruppi principali:

- 1) Addr: è l'indirizzo della potenziale successiva microistruzione.
- 2) Jam: determina come viene selezionata la prossima microistruzione.
- 3) ALU: seleziona le funzioni dell'ALU e dello shifter.
- 4) C: selezione quali registri sono scritti dal bus C.
- 5) Mem: seleziona la funzione in memoria.
- 6) B: seleziona quale registro è scritto sul bus B.

Micro Architettura Mic-1

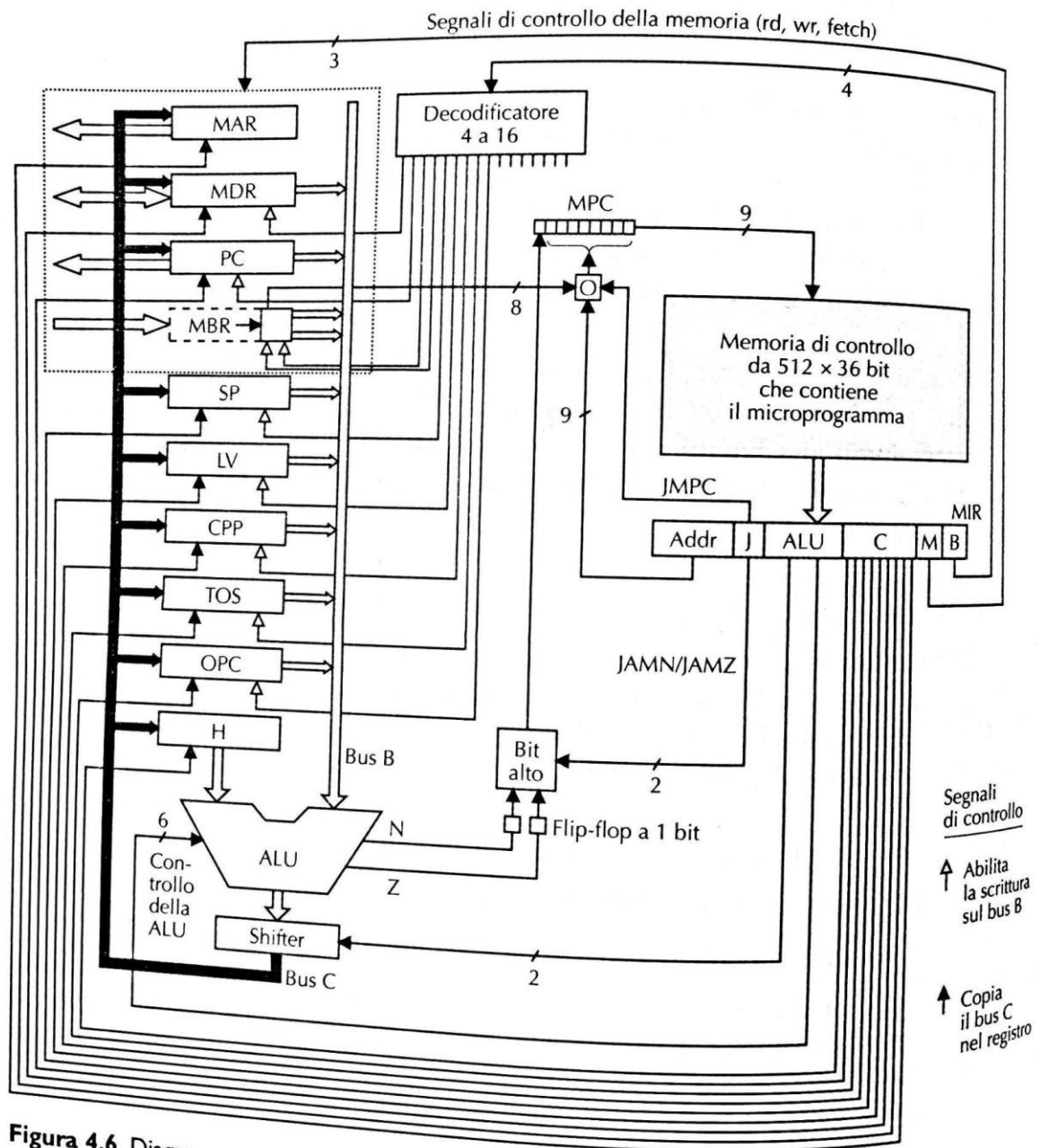


Figura 4.6 Diagramma a blocchi del nostro esempio di microarchitettura di Mic-1.

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.

Come si può osservare nella figura sopra, la parte più grande e importante della sezione di controllo è una memoria chiamata “memoria di controllo” che memorizza le microistruzioni. Il “decoder” decodifica il tipo di istruzione in base al opcode (codice del tipo di operazione da svolgere).

Il funzionamento di Mic-1

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).

Il livello ISA e un esempio di ISA: IJVM

Il **livello ISA** (Instruction Set Architecture) 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. Il livello ISA è l'interfaccia tra i compilatori e l'hardware quindi il linguaggio che entrambi possono comprendere.

Questo livello deve essere retrocompatibile.

Normalmente ha due modalità operative:

1. Modalità kernel - Per eseguire il SO e tutte le istruzioni.
2. Modalità utente - Per eseguire i programmi utente e non operazioni “sensibili” (come quelle che accedono alla cache).

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.

- Lo stack
- Il modello della memoria
- L'insieme delle istruzioni

Ortogonalità

L'ortogonalità è il principio secondo cui ogni istruzione dovrebbe essere in grado di utilizzare qualsiasi modalità di indirizzamento supportata. Fornendo una varietà di modalità di indirizzamento, 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. L'ortogonalità veniva spesso descritta come altamente "un po' efficiente".

Lo stack

E' una struttura dati utilizzata per memorizzare lo stato di una procedura che segue la filosofia LIFO (Last-In-First-Out).

Questa organizzazione dei dati permette di gestire anche situazioni di chiamate ricorsive (procedure che richiamano se stesse).

Il modello della memoria JVM

La memoria può essere vista come un vettore di:

- 4.294.967.296 Byte (4 GB)
- 1.073.741.824 parole di 4 Byte

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

1. *Porzione costante di memoria.* I programmi JVM 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.
2. *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.
3. *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.
4. *Area dei metodi.* Infine c'è una regione di memoria in cui risiede il programma. Qui è presente un registro implicito che contiene l'indirizzo della successiva istruzione da prelevare. Questo puntatore è chiamato contatore di programma (Program Counter), oppure PC.

Tutti i computer suddividono la memoria in celle adiacenti di un byte che sono a loro volta raggruppati in gruppi di 4 (32 bit) o 8 (64 bit).

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 malware).

Insieme delle istruzioni della JVM

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 <i>W 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

I Registri

Tutti i computer hanno dei registri visibili a livello ISA (alcuni di questi non 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.
- 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.

Intel Core i7

Overview 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).

Modalità operative di Intel Core i7

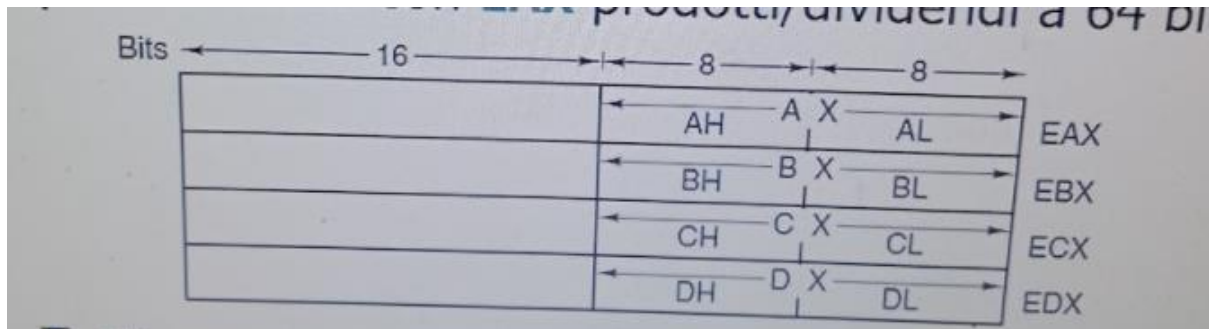
Il Core i7 ha 3 diverse modalità operative:

1. Modalità reale: si comporta come un 8088 ma nel caso vengono eseguite istruzioni errate la macchina va in blocco.
2. Modalità virtuale: permette di eseguire programmi 8088 in modo protetto e controllato da un vero SO.
3. 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.

Registri di Intel Core i7

Il core i7 ha 4 registri di uso generale (tutti a 32 bit):

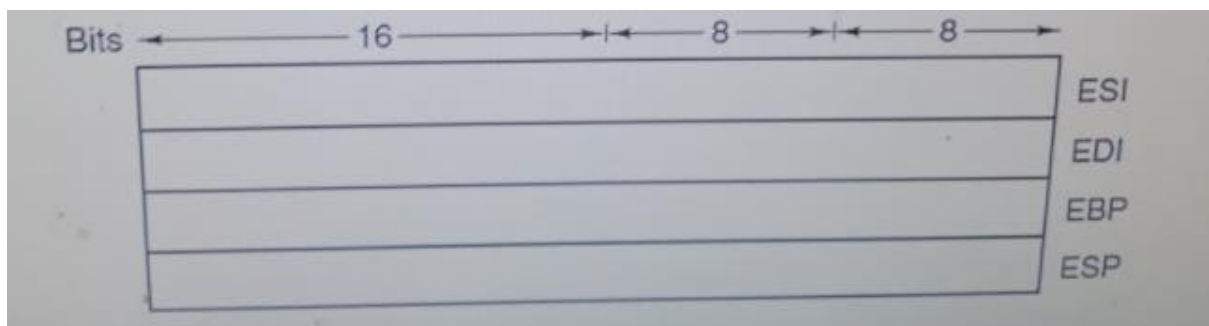
- EAX: usato per le operazioni aritmetiche.
- EBX: usato come puntatore a indirizzi di memoria.
- ECX: usato come contatore nei cicli.
- EDX: usato nelle moltiplicazioni/divisioni durante le quali contiene con EAX prodotti/dividendi a 64 bit.



Tutti questi registri possono essere utilizzati a 16 o 8 bit.

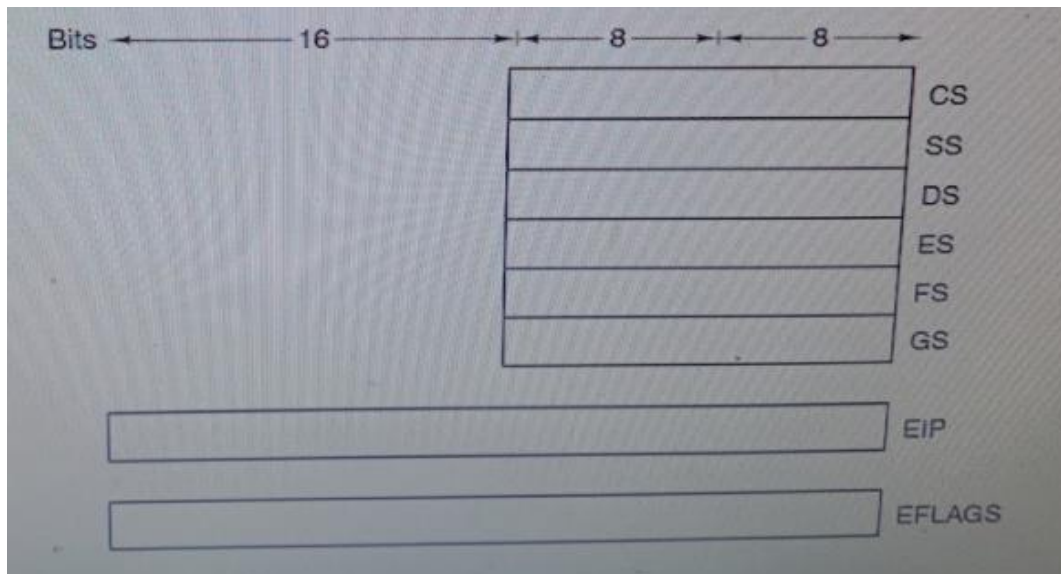
Ulteriori 4 registri a 32 bit con caratteristiche specifiche sono:

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



Poi sono presenti:

- 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.



Architetture per il calcolo parallelo

L'obiettivo principale dell'industria dei computer è sempre stato quello di incrementare le performance.

In passato questo è stato possibile incrementando la frequenza del clock. Poi, grazie alla teoria della relatività, si creò un limite fisico cioè si capì che nessun segnale elettrico può propagarsi più velocemente della velocità della luce.

Un computer con un clock di 1 THz dovrebbe essere piccolissimo (100 μm). Rendere un computer di queste dimensioni è possibile ma si presenterebbe un ulteriore problema, cioè la dissipazione del calore.

Quindi per ottenere una maggiore potenza di calcolo ci si affidò alle **architetture parallele: molte CPU (con velocità normale) che collaborano per raggiungere lo stesso obiettivo.**

La classificazione si basa su due concetti: il flusso di istruzioni ed il flusso dei 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

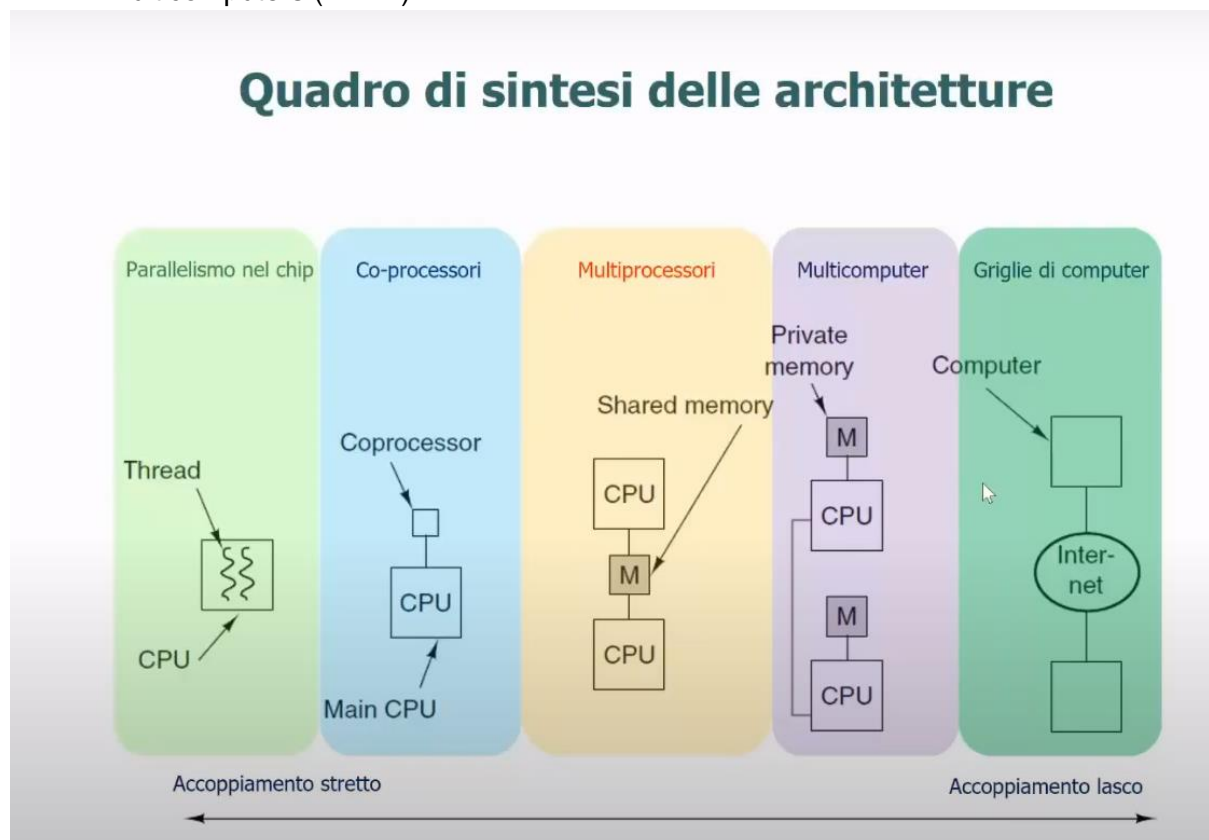
Parallelismo

Il parallelismo nel chip aiuta a migliorare le performance della CPU: con il pipelining e le architetture superscalari si può arrivare ad un fattore di miglioramento da 5 a 10.

Per incrementare drasticamente le performance di un calcolatore occorre progettare sistemi con molte CPU, in questo caso si può arrivare ad ottenere un incremento di 50, 100, o anche di più.

Esistono quindi 3 differenti approcci:

- Data Parallel Computers (SIMD)
- Multiprocessors (MIMD)
- Multicomputers (MIMD)



Parallelismo a livello delle istruzioni:

L'idea è di emettere più istruzioni per ciclo di clock utilizzando processori superscalari e processori con parole di istruzione molto lunghe.

Multithreading nel chip:

Serve per superare lo stallo che si verifica nelle pipeline quando si accede ad un indirizzo assente nelle cache.

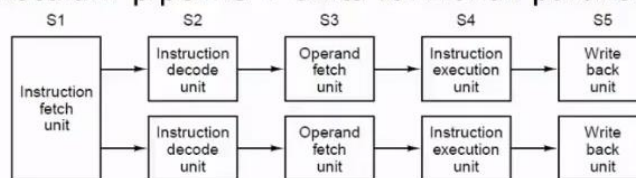
Multiprocessori in un solo chip:

L'idea sarebbe più core sullo stesso chip che condividono le cache e la memoria principale.

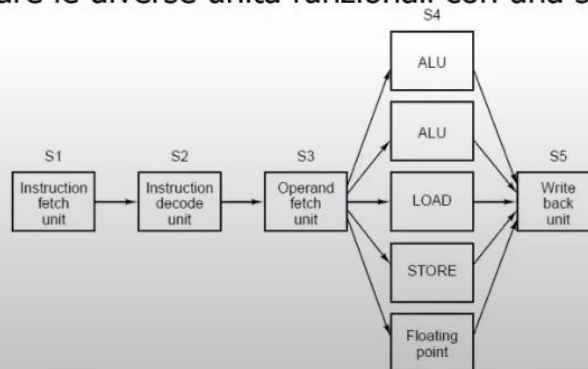
Parallelismo a livello delle istruzioni

A livello delle istruzioni, un modo per ottenere il parallelismo è emettere più istruzioni per ciclo di clock. Ci sono due tipi di CPU a emissione multipla: i processori superscalari e i processori VLIW.

- CPU superscalari: pipeline + unità funzionali parallele



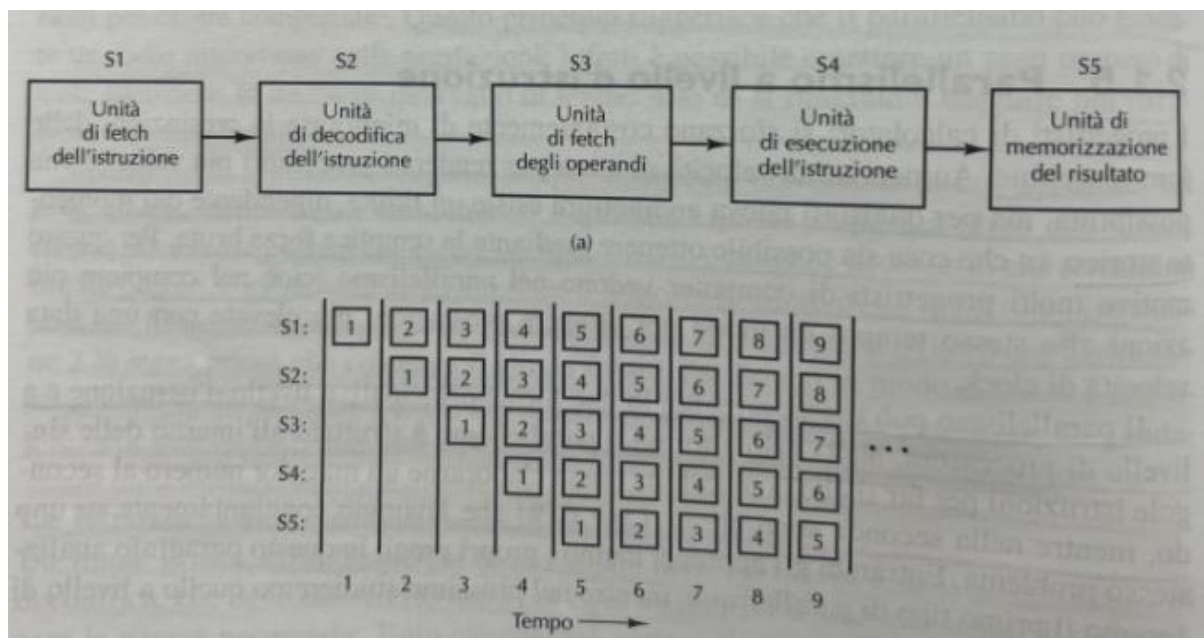
- Processori VLIW (Very Long Instruction Word) in grado di indirizzare le diverse unità funzionali con una sola linea di pipeline



Pipelining

Da anni ormai è assodato che uno dei maggiori colli di bottiglia nella velocità di esecuzione delle istruzioni è rappresentato dal prelievo delle istruzioni dalla memoria. Fin dal modello IBM Stretch (1959), per alleviare questo problema, i calcolatori sono stati dotati della capacità di poter prelevare in anticipo le istruzioni dalla memoria, in modo da averle già a disposizione nel momento in cui dovessero rendersi necessarie. Le istruzioni venivano memorizzate in un insieme di registri chiamati buffer di prefetch, dai quali potevano essere prese nel momento in cui venivano richieste, senza dover attendere che si completasse una lettura della memoria. In pratica la tecnica di prefetching divide l'esecuzione dell'istruzione in due parti: il prelievo dell'istruzione e la sua esecuzione effettiva. **Il concetto di pipeline spinge questa strategia molto più avanti; invece di dividere l'esecuzione di un'istruzione solamente in due fasi, la si divide in un numero maggiore di parti (spesso una dozzina o più) che possono essere eseguite in parallelo; ciascuna di queste parti è gestita da componenti hardware dedicati.** Viene quindi illustrato il modello di pipeline. Durante il primo ciclo di clock lo stadio S1 sta lavorando sull'istruzione 1,

prelevandola dalla memoria. Durante il ciclo di clock 2 lo stadio S2 decodifica l'istruzione 1, mentre lo stadio S1 preleva l'istruzione 2. Durante il ciclo 3 lo stadio S3 preleva gli operandi per l'istruzione 1, lo stadio S2 decodifica l'istruzione 2 e lo stadio S1 preleva la terza istruzione. Durante il quarto ciclo lo stadio S4 esegue l'istruzione 1, S3 preleva gli operandi per l'istruzione 2, S2 decodifica l'istruzione 3 e S1 preleva l'istruzione 4. Infine, durante l'ultimo ciclo S5 scrive il risultato dell'istruzione 1, mentre gli altri componenti lavorano sulle istruzioni successive. Per rendere più chiaro il concetto di pipelining consideriamo un'analogia. Immaginiamo una fabbrica di dolci in cui i reparti di cottura e di confezionamento sono separati.



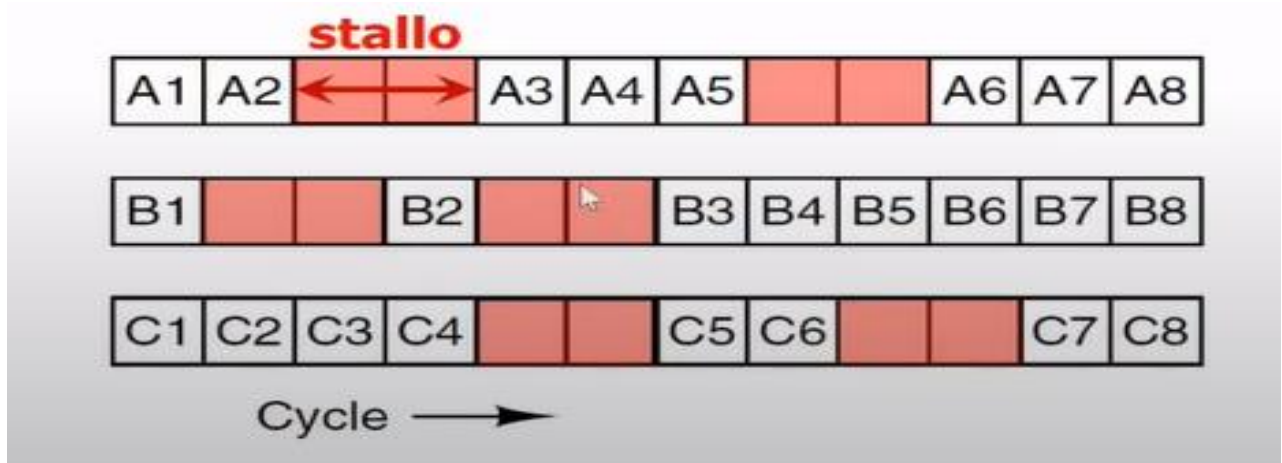
Il problema dello stallo della pipeline

Questo problema si verifica quando una CPU tenta di accedere ad un riferimento in memoria che non è nella cache, quindi deve attendere il caricamento prima di riprendere l'esecuzione.

Il Multithreading nel chip permette di mascherare queste situazioni attraverso lo switch tra thread, esistono differenti approcci:

- Multithreading a grana fine
- Multithreading a grana grossa
- Multithreading simultaneo

Supponiamo di avere una CPU che emette una 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 un riferimento non presente nella cache di primo livello e deve attendere 2 cicli per recuperarlo dalla cache di secondo livello.



Multithreading

indica il supporto hardware da parte di un processore di eseguire più thread. Questa tecnica si distingue da quella alla base dei sistemi multiprocessore per il fatto che i singoli thread condividono lo stesso spazio di indirizzamento, la stessa cache e lo stesso translation lookaside buffer. Il multithreading migliora le prestazioni dei programmi solamente quando questi sono stati sviluppati suddividendo il carico di lavoro su più thread che possono essere eseguiti in parallelo

Tutte le moderne CPU a pipeline presentano un problema: quando un riferimento in memoria fallisce nella cache di primo e di secondo livello, bisogna aspettare molto tempo prima che la parola richiesta sia caricata nella cache, e così nel mentre la pipeline si trova in stato di stallo.

Il Multithreading nel chip costituisce un modo per trattare questa situazione perché, in una certa misura, si riescono a mascherare questi stalli consentendo alla CPU di gestire contemporaneamente più thread di controllo.

Esistono diverse varianti di Multithreading...

Multithreading a grana fine

Sono eseguite ogni ciclo di clock, a turno, le singole istruzioni dei thread. Nei due cicli di stallo la CPU è occupata a svolgere le istruzioni degli altri due thread. Poiché non c'è alcuna relazione tra i thread, ciascuno ha il proprio insieme di registri. Il numero massimo di thread concorrenti è definito a priori in fase di progettazione del chip.

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

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

Multithreading a grana grossa

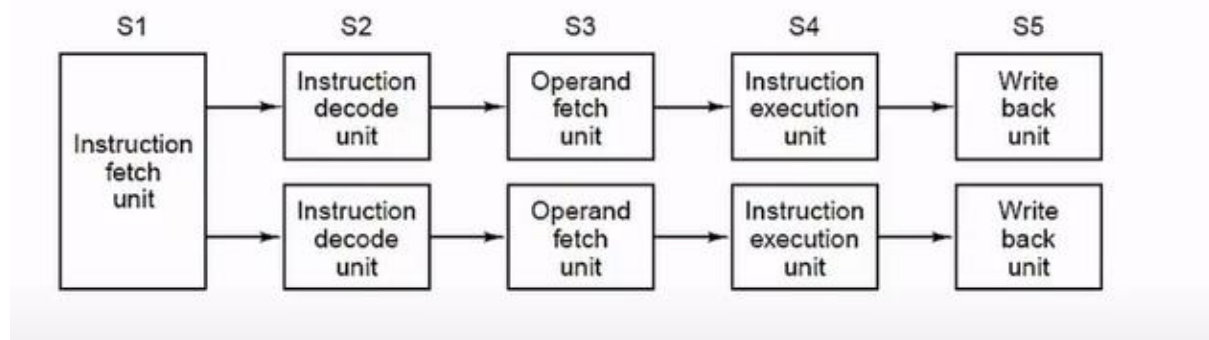
Un thread va avanti finché non raggiunge uno stallo, perde un ciclo e commuta su un altro thread. Perde un ciclo ogni stallo ed è inefficiente rispetto al multithreading a grana fine.

Richiede meno thread per mantenere occupata la CPU.

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

Multithreading a CPU superscalari

Le CPU possono avere più unità in parallelo ed emettere più istruzioni per ciclo.



In ogni thread sono eseguite due istruzioni per ciclo finché non si raggiunge uno stallo:

A1	A2			A3	A4	A5			A6	A7	A8
----	----	--	--	----	----	----	--	--	----	----	----

Esempio iniziale

B1			B2			B3	B4	B5	B6	B7	B8
----	--	--	----	--	--	----	----	----	----	----	----

C1	C2	C3	C4			C5	C6			C7	C8
----	----	----	----	--	--	----	----	--	--	----	----

Cycle →

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

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

Ciascun thread emette due istruzioni per ciclo finché non si raggiunge uno stallo.
A questo punto si passa all'istruzione del thread che segue affinché la CPU rimanga impiegata.

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.

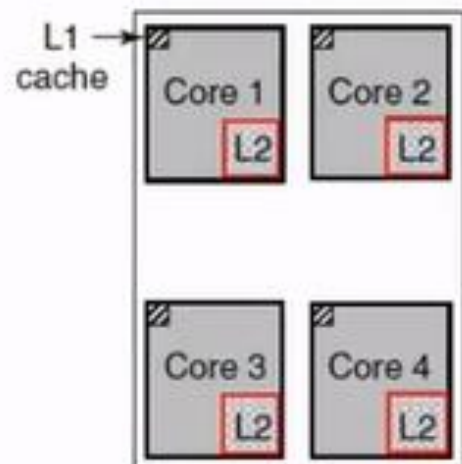
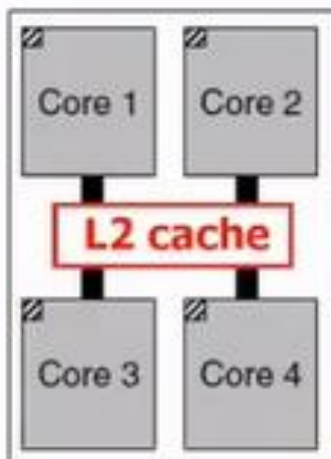
A causa del raggiungimento dei limiti fisici dei materiali non è stato possibile incrementare la frequenza di clock per ottenere CPU veloci. Per tale ragione le industrie produttrici di chip hanno incominciato ad inserire più CPU (core) all'interno dello stesso chip.

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 contengono più processori e che condividono la cache e la memoria principale sono dette multiprocessori. Esistono due tecnologie di multiprocessori in un solo chip:

- 1) 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.
- 2) Quelle che hanno più CPU ciascuna con la propria pipeline. L'interazione tra CPU non è semplice poiché risultano maggiormente disaccoppiate.



Mentre le CPU possono o meno condividere la 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 consistenza.

Multiprocessori eterogenei in un solo chip

Oltre ai multicore simmetrici esiste un altro tipo di chip multicore in cui ogni core ha un compito specifico.

- poiché queste architetture realizzano un vero e proprio calcolatore completo in un singolo chip sono spesso dette system on a chip.

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

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

Chip-level Multiprocessor

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

Dal punto di vista software essi non sono così 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 "ingordito" satura la cache L2. Altra differenza rispetto ai multiprocessori è la minore tolleranza ai malfunzionamenti causati dalla stretta connessione dei core.

Coprocessori

E' possibile velocizzare il calcolatore mediante l'aggiunta di un secondo processore specializzato chiamato "Coprocessore". Il coprocessore permette di migliorare le performance del calcolatore.

Esistono diverse varianti di coprocessori:

- Processori di rete.
- Processori grafici.
- Crittprocessori.

MIMD

Tutte le comunicazioni tra componenti elettronici 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.

Caching

Il caching è il processo di memorizzazione dei dati in un luogo separato (chiamato cache) in modo tale che sia possibile accedervi più rapidamente se gli stessi dati vengono richiesti in futuro. Quando vengono richiesti alcuni dati, la cache viene prima controllata per vedere se contiene quei dati. Se i dati sono già nella cache, la richiesta può essere soddisfatta più velocemente.

Multiprocessori e Multicomputer (MIMD)

Si possono aumentare ulteriormente le prestazioni delle macchine con la combinazione di più CPU per formare sistemi più grandi. Questi sistemi si dividono in multiprocessori e multicomputer.

Definizione Multiprocessore:

Un multiprocessore è un calcolatore in cui tutte le CPU condividono una memoria comune. Tutti i processi che cooperano in un multiprocessore possono condividere un solo spazio degli indirizzi virtuali mappato nella memoria comune.

Definizione Multicomputer:

Un Multicomputer è un'architettura parallela che prevede per ogni CPU una memoria virtuale privata, accessibile solo da essa e non dalle altre. Questo è l'aspetto che distingue un multicomputer da un multiprocessore.

Ci sono vari tipi di multiprocessori e multicomputer:

- 1) Multiprocessori a memoria condivisa: meno di 1000 CPU comunicano attraverso una memoria condivisa; il tempo di accesso ad una parola 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 μ s; 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.

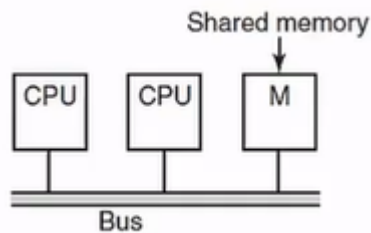
Hardware dei multiprocessori (UMA e NUMA)

Esistono due differenti tipologie di multiprocessori:

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

UMA con architettura basata su bus

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

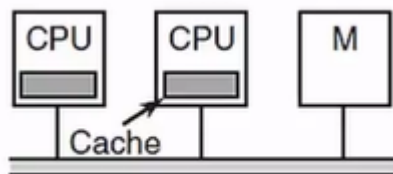


Una CPU che vuole leggere/scrivere una parola in memoria, se il bus è occupato, deve attendere che si liberi. Questa architettura funziona bene con due o tre CPU.

Per poter incrementare il numero di CPU, e quindi le performance, occorre aggiungere 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 e il sistema può supportare anche più di 3 CPU.



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 oppure READ/WRITE.

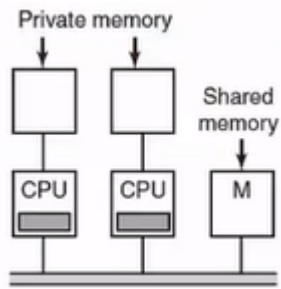
Se una CPU scrive una parola che è contenuta anche in altri blocchi di cache remote, l'hardware del bus percepisce la scrittura e lo segnala alle altre cache che possono avere:

- una copia "pulita" del blocco di memoria, allora la cache la sovrascrive con il valore aggiunto
- una copia modificata del blocco di memoria (detta "copia sporca"), 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

Un'altra soluzione è con l'aggiunta di memorie RAM in un bus interno dedicato per ogni CPU.

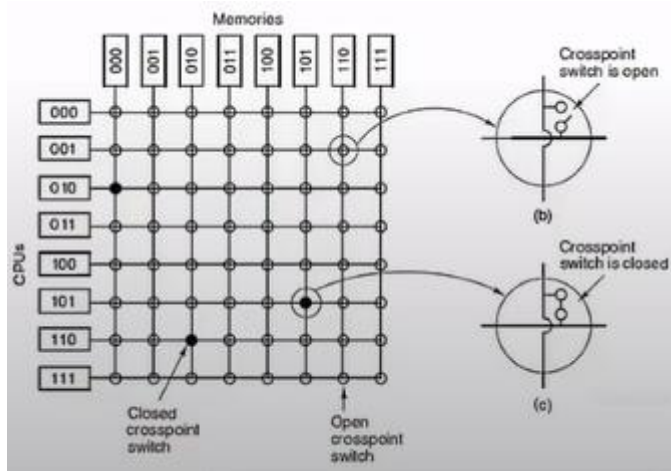


A questo punto la memoria condivisa è utilizzata esclusivamente per scrivere variabili condivise (globali). Questa soluzione riduce il traffico sul bus ma richiede una collaborazione attiva del compilatore che deve separare gli oggetti locali da quelli globali.

UMA con crossbar switch

Anche con le migliori tecniche di caching, l'uso di un singolo bus di interconnessione limita la dimensione del multiprocessore UMA a 16 o 32 CPU.

Per andare oltre occorre utilizzare una differente rete di interconnessione. Il circuito più semplice che permette di collegare n CPU a k memorie è il crossbar switch.



In ogni intersezione delle linee orizzontali (CPU) con quelle verticali (RAM) c'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 qualche crosspoint o linea è già occupata.

Non è necessaria alcuna azione di pianificazione anticipata. E' sempre possibile connettere 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.

Una delle peggiori caratteristiche di questo schema è che il numero degli incroci cresce come n^2 .

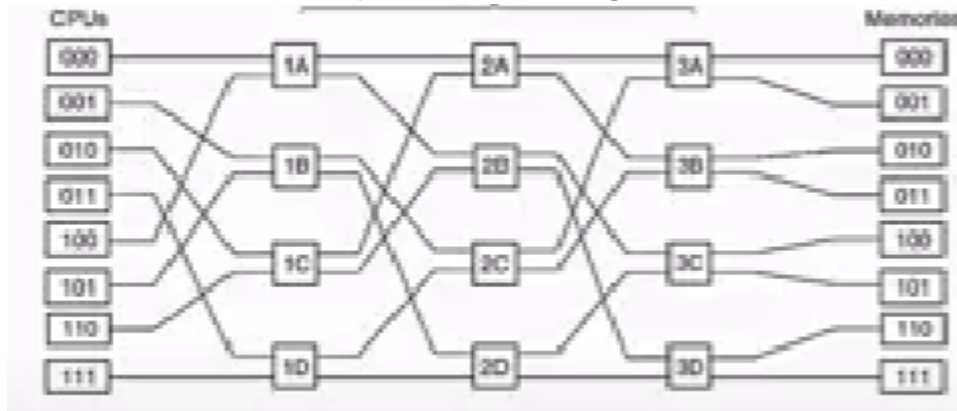
- Con 1000 CPUs e 1000 moduli di memoria occorrono un milione di crosspoints. Costruire una crossbar di queste dimensioni non è fattibile.

UMA con rete di commutazione a più stadi (o livelli)

Una progettazione di multiprocessori completamente differente è basata su switch 2x2 (due input e due output), i messaggi che arrivano nelle due linee di ingresso possono scambiarsi in una delle due linee in uscita. Ogni messaggio contiene: quale memoria utilizzare, l'indirizzo del modulo, il codice dell'operazione e il valore di un operando.

Lo switch utilizza il campo Module per scegliere dove spedire il messaggio.

Una rete economica e semplice è la rete **Omega**



sono state connesse 8 cpu ad 8 memorie con 12 switch. Per n CPU e n memorie sono necessari $\log_2 n$ stadi per un totale di $(n/2)\log_2 n$ switch.

Lo schema di connessione è detto shuffle perfetto. A differenza del crossbar la omega è una rete bloccante, non tutti gli insiemi di richieste possono essere processati contemporaneamente.

Multiprocessori NUMA

Forniscono un singolo spazio degli indirizzi a tutte le CPU.

In un processore NUMA il tempo di accesso ai moduli di memoria locale è minore rispetto a quello dei moduli remoti, tutti i programmi UMA girano in quelli NUMA, ma avranno performance degradate.

Le macchine NUMA hanno 3 caratteristiche chiave :

1. c'è un singolo spazio degli indirizzi visibile a tutte le CPU
2. l'accesso in memoria remota avviene tramite istruzioni LOAD e STORE
3. l'accesso alla memoria remota è più lento rispetto all'accesso in memoria locale

quando il tempo di accesso in memoria è nascosto, si parla di **NC-NUMA(Non-Cache NUMA)**; mentre quando sono presenti cache coerenti, si parla di **CC-NUMA(Cache-Coherent NUMA)**

l'approccio migliore per la creazione di grandi sistemi multiprocessore CC-NUMA è il **multiprocessore basato su directory**.

L'idea che sta alla base è quella di mantenere un DB che indica dove sia ciascuna linea di cache e quale sia il suo stato; quando si ha un riferimento ad una linea di cache, si interroga il DB perchè trovi dov'è e se è pulita o sporca(modificata). Occorre mantenere il DB in un hardware veloce.

Tipi di SO multiprocessore

esistono diverse organizzazioni. le più usate sono :

1. ogni CPU ha il proprio SO
2. multiprocessori Master-Slave
3. multiprocessori simmetrici

Ogni CPU ha il SO personale

in questa organizzazione, la memoria è divisa in partizioni, tante quante sono le CPU, fornendo ad ogni CPU la propria memoria virtuale e il proprio SO

bisogna ricordare 4 aspetti fondamentali di questa organizzazione:

1. quando un processo fa una system call, essa viene intercettata dalla propria CPU, utilizzando le proprie strutture dati;
2. ogni CPU ha il proprio insieme di processi, che schedula per conto proprio. Non c'è condivisione dei processi, e può capitare che se la CPU 1 è inattiva, la CPU 2 è carica di lavoro (non c'è **bilanciamento del carico**);
3. non c'è condivisione delle pagine, e non c'è alcun modo di ottenere pagine da una CPU all'altra;
4. se il sistema operativo mantiene una cache dei blocchi del disco recentemente usati, ciascun SO lo farà in modo indipendente; è possibile che un certo blocco sia presente e "sporco" in più CPU contemporaneamente. Per eliminare questo problema basterebbe eliminare la cache, ma ciò porterebbe ad un abbassamento delle prestazioni.

Multiprocessore Master-Slave

In questa organizzazione, una sola CPU contiene il SO e le tabelle delle pagine, mentre le altre eseguono processi utente questa organizzazione viene chiamata **master-slave** dato che la CPU 1 è il padrone (master) e le altre gli schiavi (slave) questo modello risolve la maggior parte dei problemi del modello precedente; infatti non accade mai che una CPU sia inattiva e un'altra carica di lavoro c'è anche uno svantaggio: con tanti slave, il master potrebbe diventare il collo di bottiglia della struttura, perchè dovrebbe gestire tutte le system call da parte degli slave.

Multiprocessori Simmetrici

Gli **SMP(Symmetric MultiProcessor)** eliminano questa asimmetria: in memoria è presente una copia del SO, ma ogni CPU può eseguirlo. La CPU che ha eseguito la system call, effettua una trap al kernel ed elabora la richiesta.

Ogni CPU può diventare il master della struttura questo modello bilancia i processi e la memoria dinamicamente. elimina il collo di bottiglia della CPU master, ma ha qualche svantaggio: se due o più CPU eseguono codice del SO in contemporanea, avverrà un disastro.

Per evitare questi problemi, il modo più semplice è associare un mutex al SO,rendendo questo una grande regione critica. Quando una CPU vuole eseguire codice del SO, deve prima acquisire il mutex:se è occupato, la CPU aspetterà.

Sincronizzazione dei multiprocessori

le CPU dei multiprocessori vanno sincronizzate di frequente.

Per essere sincronizzate,occorrono primitive di sincronizzazione appropriate tutte le CPU devono usare e rispettare un protocollo mutex appropriato, per garantire la mutua esclusione il cuore di ogni mutex è l'istruzione TSL per implementare le regioni critiche in un multiprocessore, l'istruzione TSL deve bloccare il bus, poi effettuare entrambi gli accessi in memoria e infine sbloccare il bus. Il lock del bus avviene richiedendolo al bus stesso, per mezzo del protocollo ordinario di richiesta, quindi ponendo a livello logico alto(cioè ad 1) una linea speciale del bus.

Se TSL è implementata bene, la mutua esclusione viene sempre garantita. Questo metodo viene chiamato **spin lock**.

Scheduling dei multiprocessori

in un multiprocessore lo scheduling è bidirezionale:lo scheduler deve scegliere quale processo schedulare e su quale CPU.

Ci sono 3 tipi di scheduling:

1. Timesharing
2. Condivisione dello spazio
3. Schedulazione Gang

Timesharing

l'algoritmo di scheduling a timesharing viene usato solo quando i processi sono indipendenti, utilizza un vettore di liste e thread(tutti nello stato ready) a diverse priorità di esecuzione questa schedulazione con una sola struttura dati, utilizzata da tutte le CPU, ripartisce il tempo tra le CPU come se ci trovassimo in un multiprocessore, fornendo anche un bilanciamento del carico automatico.

I due svantaggi sono:

1. la potenziale contesa per la struttura dati di schedulazione.
2. overhead nell'effettuare il context-switch tra i processi che si bloccano su I/O.

Se un processo tiene un lock gestito da ciclo di attesa, le altre CPU sprecano tempo per ciclare in attesa che il processo rilasci il lock per ovviare a questo problema si usa lo **smart scheduling**, dove un processo, con un lock come prima,utilizza un flag a livello di processo per evidenziare che detiene correntemente il lock;quando rilascia il lock,azzerà il flag. Lo scheduler non ferma un processo che detiene il lock, ma gli concede un po 'di tempo in più.

Alcuni multiprocessori utilizzano lo **scheduling per affinità**: l'idea di base è quella di eseguire un processo sulla stessa CPU dove è stato eseguito l'ultima volta.

un modo per creare ciò è la **schedulazione a due livelli**: quando un processo viene creato, viene assegnato ad una CPU, e tale assegnazione costituisce il livello superiore dell'algoritmo

come risultato otteniamo che ogni CPU acquisisce il proprio insieme di processi, ci sono 3 vantaggi:

1. **il carico tra le CPU è distribuito in modo paritario**
2. **sfrutta l'affinità della cache**
3. **viene minimizzata la contesa per le liste**

Condivisione dello spazio(Space Sharing)

Questo modello si può utilizzare quando i processi sono correlati fra di loro

questo modello viene chiamato **space sharing**, funziona così:

supponiamo di creare un insieme di thread contemporaneamente; in quel momento lo scheduler controlla se ci sono tante CPU libere quanti sono i thread se esistono, assegnano ad ogni thread la sua CPU dedicata, e partono tutti insieme; se non ci sono abbastanza CPU, non si fa partire nessun thread, finché non è disponibile un numero di CPU adeguato. In ogni istante, l'insieme delle CPU è suddiviso in un certo numero di partizioni, ciascuna delle quali esegue i thread di un processo.

Un vantaggio dello space sharing è che elimina l'overhead dovuto al cambio di contesto.

Uno svantaggio è il tempo perso quando una CPU si blocca.

Schedulazione Gang

Una soluzione ai problemi dello space sharing è la **schedulazione gang, uno sviluppo della co-schedulazione**. è costituita da 3 parti:

1. gruppi di thread correlati sono schedulati come un'unità;
2. tutti i membri di una gang sono in esecuzione contemporaneamente, in differenti CPU in time sharing;
3. tutti i membri di una gang iniziano e finiscono nello stesso istante.

Questo modello funziona perché tutte le CPU sono schedulate in sincronia: questo significa che il tempo è suddiviso in quanti discreti.

All'inizio di un nuovo quanto, tutte le CPU sono di nuovo schedulate, e un nuovo thread inizia su ciascuna di esse.

L'idea di questo modello è quella di avere tutti i thread di un processo in esecuzione in contemporanea, cosicché se uno di essi manda una richiesta all'altro, quest'ultimo riceverà il messaggio e sarà in grado di rispondere quasi immediatamente.

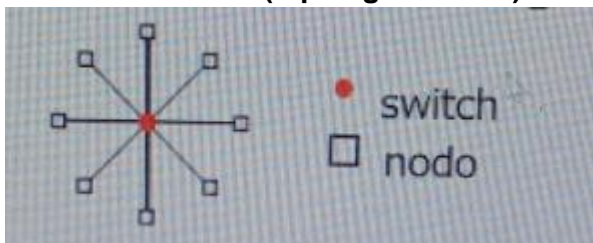
Multicomputer

- 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.

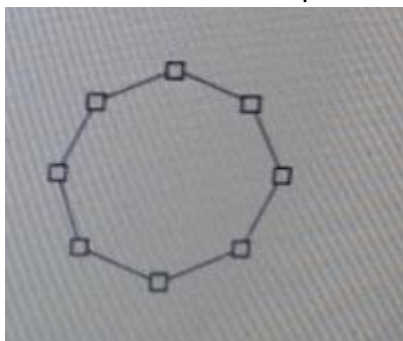
- I multiprocessori di grandi dimensioni sono difficili da costruire e perciò risultano costosi.
- Per ovviare a questi problemi sono nati i multicomputer che sono fortemente accoppiati ma non condividono memoria.
- Ogni computer ha una propria memoria.
- Questi sistemi sono anche chiamati Cluster di Computers oppure Cluster di Workstations.
- I multicomputer sono facili da costruire perché il componente base è un PC con una scheda di rete con alte performance.
- Il segreto per ottenere alte performance in un multicomputer è nel progetto della rete di interconnessione e delle schede di rete.
- I messaggi sono spediti in un tempo nell'ordine dei μsec , mille volte di 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 (Topologia)

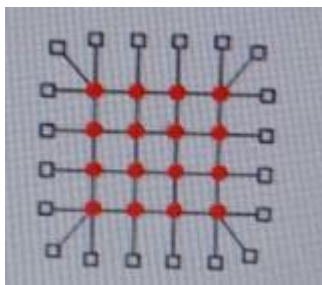
- Il nodo base di un multicomputer consiste di una CPU, una memoria e una interfaccia di rete (talvolta anche un hard disk).
- Ogni nodo ha una interfaccia di rete con uno o due cavi (o fibra) che lo connette agli altri nodi oppure agli switch.
- In un piccolo sistema, ci potrebbe essere uno switch attraverso il quale sono connessi tutti i nodi (**topologia a stella**)



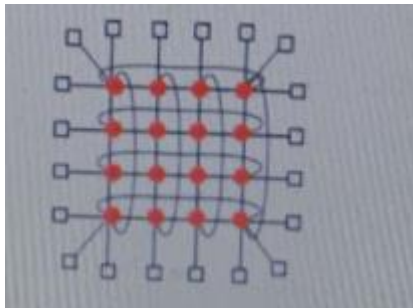
- Un'altra rete di interconnessione è la topologia ad **anello**: ogni nodo è connesso ad altri due nodi in ordine per formare un anello circolare (non sono necessari switch).



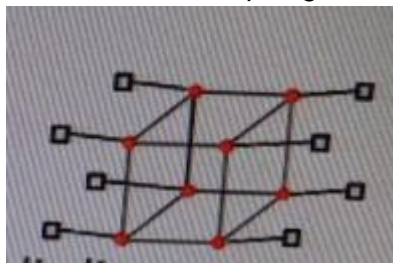
- La **griglia (grid) o maglia (mesh)** è una struttura bidimensionale utilizzata in molti sistemi complessi. La sua forma regolare la rende altamente scalabile. Il percorso più lungo tra due nodi, **diametro**, aumenta come la radice quadrata del numero dei nodi.



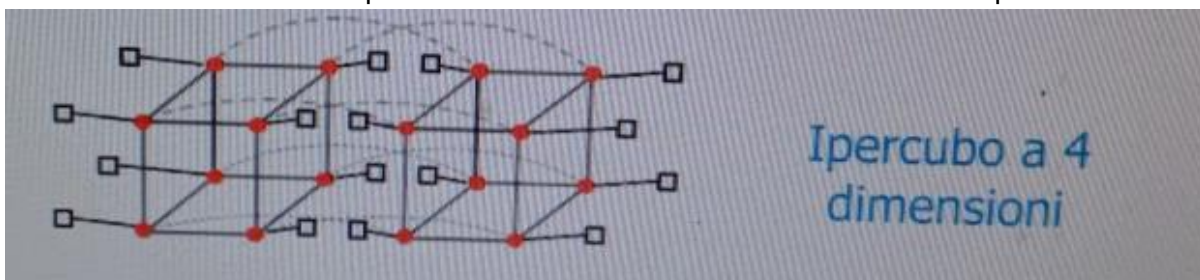
- Il **doppio toro** è una variante della griglia con i nodi estremi che si congiungono, è più tollerante ai guasti ma ha un diametro più piccolo.



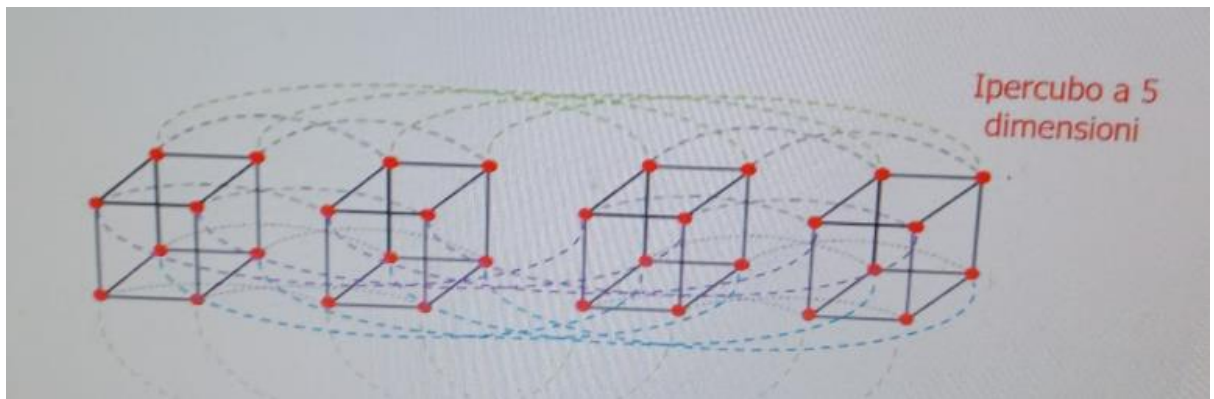
- Il **cubo** è una topologia tridimensionale regolare.



- Un cubo di dimensione 4 si può ottenere da uno di dimensione 3 ed è detto ipercubo.



- Molti computer paralleli utilizzano questa topografia perché il diametro cresce in modo lineare con la dimensione.
 $\text{diametro} = \log_2 \text{numero di nodi}$.
- Un cubo di dimensione $n + 1$ si può ottenere da uno di dimensione n in modo ricorsivo.



- Nei multicomputer sono usati due tipi di schemi di switching:
 1. Store-and-forward packet switching (connection-less).
 2. Circuit switching (connection oriented).

Store-and-forward packet switching

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 (i.e. traffico dati, priorità,...)

- E' flessibile ed efficiente ma ha il problema dell'incremento dei tempi di latenza lungo la rete di interconnessione.

Circuit switching

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

- 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** (instradamento verso la tana del verme), spezza il pacchetto in sottopacchetti e permette a quest'ultimi di iniziare il tragitto prima che sia stato inizializzato il collegamento.

Interfacce di rete

tutti i nodi di un multicomputer hanno una scheda di rete per consentire la connessione del nodo alla rete di interconnessione.

la scheda di interfaccia contiene della RAM per conservare i pacchetti in ingresso/uscita, che vanno copiati nella RAM della scheda di interfaccia, prima di poter essere trasmessi al primo switch

La scheda di interfaccia può avere uno o più canali DMA, o anche una CPU completa (**Processore di rete**). I canali DMA possono copiare i pacchetti fra la scheda di interfaccia e la RAM principale ad alta velocità, trasferendo quindi diverse parole senza dover richiedere il bus per ogni parola

Software di comunicazione a basso livello

il nemico peggiore della comunicazione ad alte prestazioni è l'eccesso di copia dei pacchetti: nel migliore dei casi avremo una copia dalla RAM alla scheda di interfaccia, una copia dall'interfaccia del sorgente a quella di destinazione, e una copia da lì alla RAM di destinazione, per un tot di 3 copie

per evitare questo numero di copie in eccesso, e l'abbassamento delle prestazioni, alcuni multicomputer mappano la scheda di interfaccia direttamente nello spazio utente, permettendo al processo utente di mettere i pacchetti direttamente sulla scheda, senza interpellare il kernel

ci sono però 2 problemi.

il primo problema sta nella competizione di due o più processi, concorrenti sullo stesso nodo, che vogliono spedire un pacchetto.

Una possibile soluzione è quella di mappare la scheda di interfaccia in tutti i processi che se ne servono, però occorre un meccanismo per evitare le corse critiche

il secondo problema sta nella condivisione della scheda di rete tra il kernel, che magari vuole accedere ad un file system remoto, ed il processo utente.

La soluzione ottimale è quella di utilizzare 2 schede di rete diverse per ciascuna funzione.

Una nello spazio utente per il traffico di applicazioni, l'altra nello spazio kernel per il SO

Software di comunicazione a livello utente

i processi sulle diverse CPU di un multicomputer comunicano attraverso lo scambio di messaggi; nella forma più semplice lo scambio di messaggi è visibile al processo utente

Send e Receive

i servizi di comunicazioni possono essere ridotti a 2 chiamate di sistema, una per mandare(**send**) messaggi e l'altra per leggerli(**receive**)

send(dest,&mptr)

receive(addr,&mptr)

poiché in un multicomputer il numero di CPU è noto a priori, il campo addr si compone di due parti:

1. Identificativo della CPU
2. Identificativo del processo o della porta sulla CPU selezionata

Primitive bloccanti e non bloccanti

Le primitive viste prima sono dette **bloccanti** (a volte anche **sincrone**):

quando un processo utente chiama la *send*, specifica la destinazione e il buffer da spedire a quella destinazione. Mentre il messaggio viene inviato, il processo si blocca; l'istruzione dopo la *send* non viene eseguita finché il messaggio non viene spedito.

un'alternativa a questo è rappresentato con le primitive **non bloccanti** (dette anche **asincrone**). La *send* non bloccante restituisce immediatamente il controllo al processo utente, prima che il messaggio venga spedito; il vantaggio è che il mittente può continuare ad eseguire calcolo in parallelo alla trasmissione del messaggio, anziché lasciare inattiva la CPU. La scelta tra primitive bloccanti o non bloccanti viene fatta dai progettisti del sistema. Oltre al grosso vantaggio delle primitive non bloccanti, esse presentano un grosso svantaggio: il mittente non ha idea di quando il destinatario ha ricevuto il messaggio, e quindi non sa quando può operare sul buffer contenente il messaggio senza fare danni.

Ci sono 3 possibili soluzioni.

la prima è che il kernel si crea una copia del messaggio in un buffer interno.

svantaggio : eccesso di copie

la seconda consiste nel generare un'interrupt verso il mittente quando il messaggio viene effettivamente spedito, per informarlo che il buffer è di nuovo utilizzabile.

svantaggio: la gestione degli interrupt a livello utente rende la programmazione complessa, e potrebbe portare a molte *race condition*.

la terza soluzione consiste nel marcare il buffer come copia in scrittura (**copy-on-write**), cioè marcare il buffer in sola lettura finché il messaggio non viene spedito.

svantaggio: eccesso di copie.

quindi dal punto di vista del mittente, abbiamo queste scelte:

1. *send* bloccante (CPU inattiva)
2. *send* non bloccante con copia (spreco di tempo di CPU per la copia)
3. *send* non bloccante con interrupt (difficile la programmazione)
4. *copy-on-write* (probabilità elevata di eccesso in scrittura)

in condizioni normali, la prima scelta è la migliore, soprattutto se sono disponibili thread multipli, così quando un thread si blocca con la *send*, gli altri svolgono altri lavori.

anche la *receive* può essere bloccante o non.

una *receive* bloccante sospende il mittente finché il messaggio non è arrivato; una *receive* non bloccante invece dice al kernel dove si trova il buffer e ritorna quasi immediatamente.

Si possono usare gli interrupt per segnalare che il messaggio è arrivato, ma sono difficili da programmare e anche piuttosto lente, quindi è preferibile che il ricevente effettui il *polling* per i messaggi in ingresso, utilizzando una procedura, ***poll***, che rileva se ci sono dei messaggi in attesa.

un ulteriore schema è quello di creare e utilizzare, nello spazio degli indirizzi del processo ricevente, un **thread pop-up**, appena arriva il messaggio. Dopo aver svolto il proprio lavoro, questo thread viene automaticamente distrutto.

una variante consiste nell'eseguire il codice di ricezione direttamente nel gestore delle interruzioni. Per velocizzare il tutto, il messaggio stesso contiene l'indirizzo del gestore delle interruzioni, così da poter richiamare il gestore con poche istruzioni.

il vantaggio è che non viene effettuata nessuna copia: il gestore preleva il messaggio dalla scheda di interfaccia e lo elabora al volo.

Questo metodo è detto schema **a messaggi attivi**.

Remote Procedure Call(Chiamate di procedura remote)

l'unico problema del modello a scambio di messaggi è che il paradigma base, su cui è costruito tutto il processo, è l'I/O.

per ovviare a questo problema Birrel e Nelson, hanno proposto di permettere ai programmi di chiamare procedure su altre CPU.

Quando il processo sulla macchina 1 chiama una procedura sulla macchina 2, il processo chiamante viene sospeso, e l'esecuzione della procedura avviene sulla macchina 2. L'informazione può essere trasportata attraverso i parametri, e tornare indietro attraverso i risultati della procedura.

questa tecnica è nota come **RPC(Remote Procedure Call)**

la procedura chiamante è nota come **client**, e quella chiamata come **server**

l'idea alla base è quella di far sembrare la chiamata di procedura remota il più possibile come una chiamata locale

I passi effettivi della RPC sono:

1. chiamata da parte del client al **client stub**(piccola procedura di libreria, locale)
2. impacchettamento dei parametri in un messaggio e system call per spedire il messaggio(l'impacchettamento dei messaggi è detto **marshaling**)
3. il kernel spedisce il messaggio dalla macchina client a quella server
4. il kernel passa il pacchetto in arrivo al **server stub**
5. il server stub chiama la procedura al server

La risposta fa lo stesso percorso in direzione opposta

Problemi Implementativi

esistono diversi problemi nascosti.

1. Con RPC il passaggio dei puntatori è impossibile, perchè il client ed il server hanno diversi spazi per gli indirizzi
2. questo capita con i linguaggi debolmente tipati come il C, in cui è perfettamente legale scrivere una procedura che calcola il prodotto scalare di due vettori senza che se ne conosca la dimensione. Con RPC è impossibile che il client riesca a fare il marshaling dei pacchetti, se non ne conosce la dimensione
3. non è sempre possibile dedurre i tipi di parametri
4. questo è relativo all'uso delle variabili globali. Di norma le procedure chiamante e chiamata possono comunicare con le variabili globali. Se la procedura chiamata viene spostata su una macchina remota, il codice fallirà, perchè non sono più condivise le variabili globali

DSM(Distributed Shared Memory)

molti programmatori preferiscono ancora un modello a memoria condivisa.

Si utilizza una tecnica chiamata DSM(Memoria Condivisa Distribuita).

Questa tecnica consiste nel fare avere ad ogni macchina la propria memoria e le proprie tabelle delle pagine. Quando una CPU effettua una LOAD o STORE su una pagina che non

ha, avviene una trap, che localizza la pagina e chiede alla CPU che la possiede di inviarla e spedirla sulla rete di interconnessione.

QUando arriva, viene mappata e viene fatta ripartire l'istruzione che aveva provocato il fault.

Virtualizzazione

- un sistema virtualizzato mantiene l'affidabilità di un sistema multicomputer ad un costo ridotto ed una maggiore semplificazione della manutenibilità
- un guasto sul server che gestisce le macchine virtuali produce un danno catastrofico rispetto a quello di una maggiore semplificazione della manutenibilità (la probabilità di un guasto naturale hardware è più bassa rispetto ad un guasto naturale software)

vantaggi della virtualizzazione

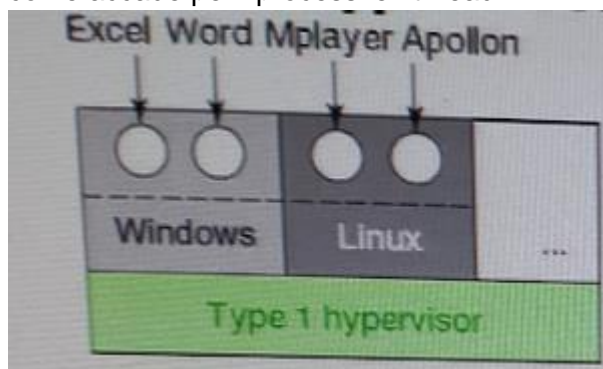
- un malfunzionamento su una macchina virtuale non inficia il comportamento delle altre
- la riduzione delle macchine fisiche riduce lo spazio, il consumo di energia, la produzione di calore quindi l'energia per il raffreddamento
- la creazione di checkpoint e la migrazione di macchine virtuali è più semplice rispetto ad un ambiente tradizionale
- si possono far girare applicazioni legacy su ambienti obsoleti che non funzionerebbero con gli attuali hardware
- i programmatori possono effettuare il test delle applicazioni su differenti SO senza disporre di hardware fisici e SO

Tipi di virtualizzazione

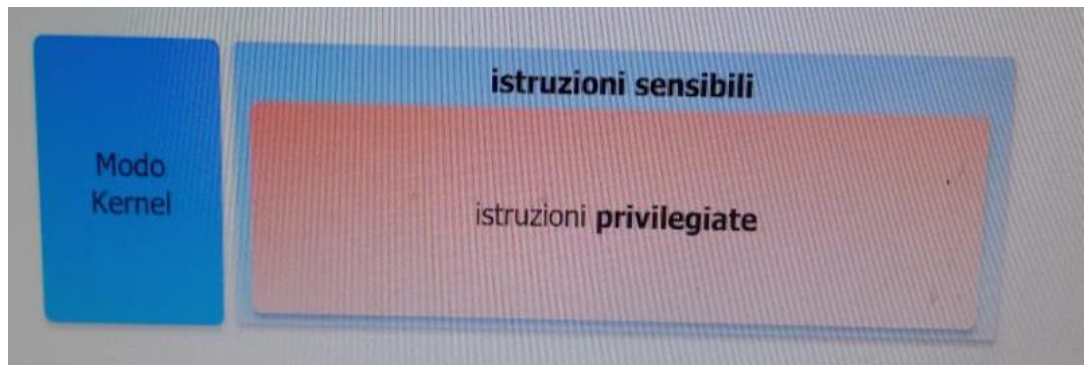
- esistono due differenti approcci per il monitor delle macchine virtuali: **hypervisor di tipo 1** e **di tipo 2**

- **tipo 1**

l'hypervisor è nel sistema operativo della macchina (**SO host**) e gira in modo kernel, il suo compito è di supportare le macchine virtuali (**SO guest**) così come accade per i processi e i thread



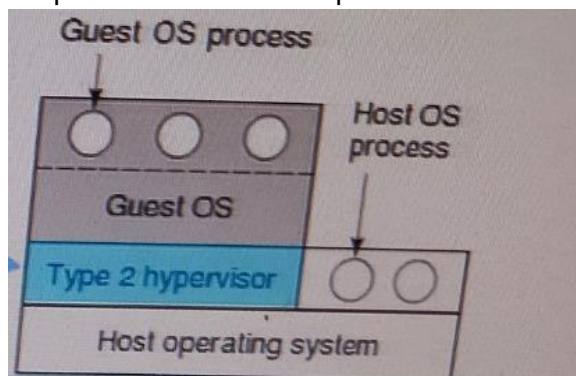
- il tipo 1 è se e solo se tutte le istruzioni sono privilegiate



- la macchina virtuale è eseguita come un processo utente in modalità utente, non possono eseguire istruzioni sensibili anche se pensa di essere in modalità kernel (**modalità virtuale kernel**)
- quando il SO guest esegue una istruzione sensibile:
 1. se la CPU non ha la VT la virtualizzazione non è possibile
 2. 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
 - un programma utente in questo caso emula il comportamento dell'hardware reale

- tipo 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.



- **VMware** è un hypervisor di tipo 2 ed è eseguito come un 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 blocchi che terminano con istruzione di flusso (es: JMP, CALL, trap,...)
 - ricerca nei blocchi le istruzioni sensibili e le traduce con una procedura **VMware**
 - il blocco è messo nella chance di **VMware** e può essere eseguita alla velocità della macchina fisica (istruzione tradotte in parte)

Confronto tra hypervisor

- negli hypervisor tipo 2 tutte le istruzioni sensibili sono sostituite da chiamate a procedura che ne emulano il comportamento. il SO guest non invierà mai nessuna istruzione sensibile alla macchina fisica
- le performance delle CPU con VT sono enormemente superiori rispetto a quelle senza VT ?
 - purtroppo l'approccio "trap-and-emulare" adottato dagli hardware VT genera troppi trap ed un eccessivo overhead di gestione, mentre la traduzione delle istruzioni sensibili è più efficiente

Paravirtualizzazione

- gli hypervisor tipo 1 e 2 funzionano senza modificare il SO guest, ma con performance meno 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 un'interfaccia 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