

Argomenti

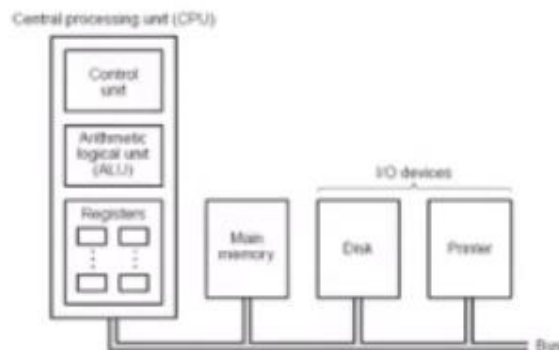
- PROCESSORI:
 - Organizzazione della CPU.
 - Esecuzione dell'istruzione.
 - RISC contro CISC.
 - Principi di progettazione dei calcolatori.
 - Parallelismo a livello di istruzione.
 - Parallelismo a livello di processore.
- MEMORIA PRINCIPALE:
 - Bit.
 - Sistemi di numerazione.
 - Indirizzi di memoria.
 - Ordinamento dei byte.
 - Codici di correzione di errore.
 - Memoria cache.

Obiettivi

- Studiare l'organizzazione della CPU.
- Comprendere le caratteristiche di progettazione delle CPU.
- Capire le attività interna dei processori.
- Comprendere i diversi sistemi di numerazione.
- Analizzare il funzionamento della memoria.
- Identificare le tecniche di correzione degli errori.

Processori

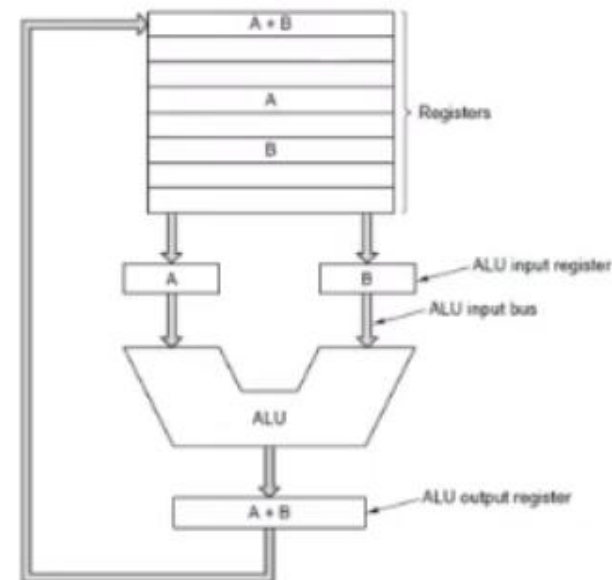
- Il cervello del computer è la CPU (Central Processing Unit).
- La CPU contiene: l'Unità di Controllo (CU), l'Unità Aritmetica e Logica (ALU) e alcuni registri (le più piccole memorie ad altissima velocità).
- Tra i registri due sono molto importanti:
 - Il **Program Counter (PC)** che "punta" alla prossima istruzione da prelevare (fetch) per l'esecuzione.
 - L'**Instruction Register (IR)** che mantiene l'istruzione corrente in fase di esecuzione.



- Le componenti di un computer sono collegate attraverso un **bus**: una collezione di cavi paralleli utilizzati per trasferire indirizzi, dati e segnali di controllo.

Organizzazione della CPU

- Una tipica CPU di von Neumann contiene il **datapath** (o percorso dati) costituito da registri (da 1 a 32), l'ALU e vari bus di collegamento così come mostrato in figura.
- L'ALU esegue, sui registri di input (A e B), addizioni, sottrazioni e altre operazioni semplici; il risultato è posto nel registro di uscita.
- Il risultato nel registro di uscita può essere memorizzato nei registri e successivamente nella memoria.
- Le operazioni possono essere:
registro-memoria (è necessaria una fase di caricamento delle parole della memoria nei registri)
register-register (gli operandi sono già pronti nei registri).



Esecuzione delle istruzioni

Il ciclo **fetch-decode-execute** è fondamentale per l'esecuzione delle istruzioni nel computer e si compone dei seguenti passaggi:

- 1) Preleva la successiva istruzione dalla memoria e la inserisce nell'IR;
- 2) Aggiorna il **program counter** in modo che punti alla prossima istruzione;
- 3) Decodifica l'istruzione in IR;
- 4) Se l'istruzione referencia una parola in memoria, la ricerca;
- 5) Se necessario preleva la parola dalla memoria e la inserisce in un registro della CPU;
- 6) Esegue l'istruzione;
- 7) Riparti dal punto 1) per eseguire la prossima istruzione.

Strategie di progettazione delle CPU

- In un'architettura **CISC** (Complex Instruction Set Computer) la CPU è in grado di comprendere molte istruzioni complesse nativamente (è il più alto livello di astrazione riconosciuto dalla macchina).
- Invece, le architetture **RISC** (Reduced Instruction Set Computer) sono basate sull'idea che se le istruzioni sono semplici e poche, esse possono essere eseguite rapidamente (è necessario un solo ciclo nel datapath!).

Un approccio ibrido:

- A partire dal x486, le CPU Intel contengono un sottoinsieme di istruzioni RISC (quelle più comuni) che possono essere eseguite in un singolo ciclo nel datapath, mentre le altre complesse sono interpretate secondo la classica modalità CISC.

Principi di progettazione dei calcolatori

I progettisti di CPU tentano di seguire un insieme di principi denominati **principi di progettazione RISC**:

- Tutte le istruzioni devono essere eseguite direttamente dall'hardware:
 - Le istruzioni non devono essere interpretate. Per le architetture CISC, quelle istruzioni più complesse (e più rare) possono essere suddivise in parti ed eseguite, successivamente, come sequenze di microistruzioni.
- Massimizzare la frequenza di emissione delle istruzioni:
 - Il parallelismo gioca un ruolo essenziale nelle prestazioni di un computer.

Principi di progettazione dei calcolatori

- Le istruzioni devono essere semplici da decodificare:
 - Le istruzioni dovrebbero essere regolari, di lunghezza predefinita, con un numero ridotto di campi/variabili.
- Soltanto istruzioni di LOAD e STORE devono fare riferimento alla memoria (che ha tempi di accesso considerevoli):
 - Tutte le altre istruzioni dovrebbero operare sui registri.
- Le CPU dovrebbero poter disporre di un elevato numero di registri:
 - Poiché 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: più istruzioni nell'unità di tempo

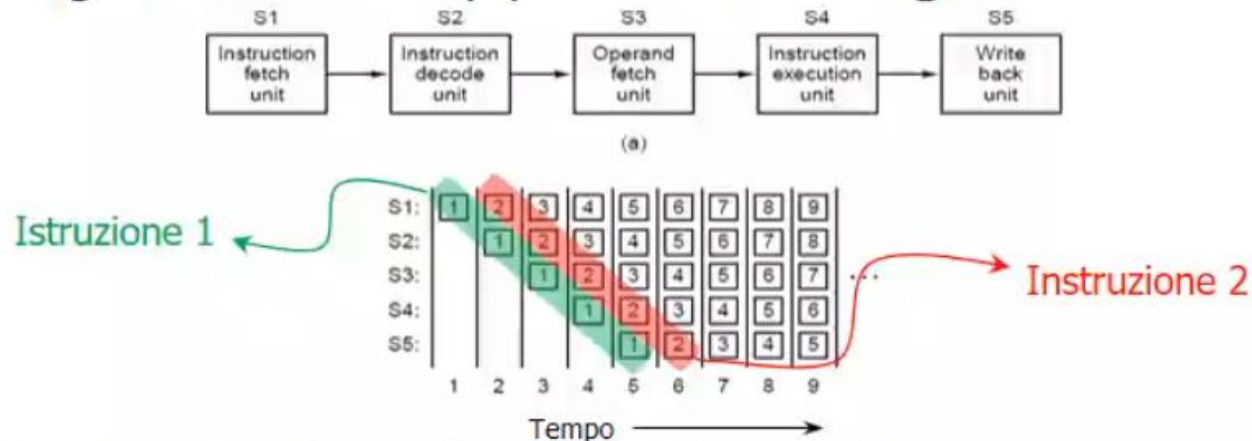
- Poiché l'incremento del clock del processore ha raggiunto un limite fisico, i progettisti di CPU guardano al parallelismo (più istruzioni nello stesso tempo) per incrementare le performance.
- Il parallelismo si può ottenere in due diversi modi:
 - **Parallelismo a livello di Istruzione.**
 - Il parallelismo 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.

Parallelismo a livello di istruzione: pipelining

- Il principale collo di bottiglia della velocità di esecuzione delle istruzioni è rappresentato dall'operazione di fetch dell'istruzione 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 **Pipeline** questa strategia è ancora più stressata: l'esecuzione dell'istruzione è divisa in molti passaggi (o stage). Queste fasi possono essere eseguite in parallelo da unità hardware dedicate.

Pipelining

- La figura mostra un pipeline con 5 **stage**:



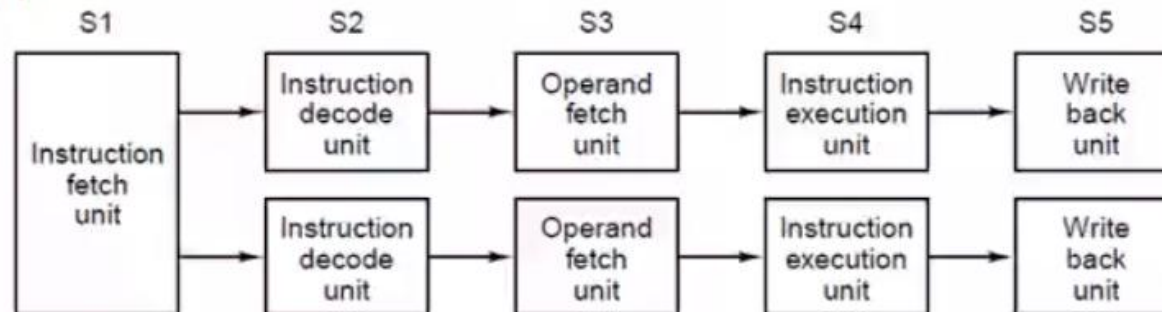
- Stage 1 Fetch dell'istruzione dalla memoria e memorizzazione nel buffer di prefetch.
- Stage 2 Decodifica dell'istruzione.
- Stage 3 Rintraccia e preleva gli operandi
- Stage 4 Esegue l'istruzione.
- Stage 5 Scrive i risultati nei registri (compreso quello di stato).

Incremento della banda del processore

- Il pipelining permette di bilanciare la **latenza** (quanto dura l'esecuzione di una istruzione) con la **banda del processore** (quanti MIPS la CPU è in grado di emettere).
- Una CPU, senza pipeline, che opera con clock di **T** ns ed emette una istruzione per ciclo di clock ha una banda di $10^3/T$ MIPS.
- Supponendo di disporre di una CPU con queste caratteristiche:
 - clock di **T** ns.
 - una pipeline a **p** stadi in cui ogni stadio necessita di un ciclo di clock per essere completato.
- La latenza totale è pari a **p·T** ns mentre la banda (cioè la velocità reale di emissione delle istruzioni) è **p·10³/T** MIPS cioè **p** volte rispetto alla banda di una CPU senza pipeline con medesima latenza: **p·10³/(p·T)**.

Processori con più pipeline

- Se una sola pipeline è buona, allora due sono sicuramente meglio!
- Questa architettura è stata utilizzata inizialmente dall'Intel x486.



- Tuttavia:
 - Non tutte le istruzioni possono essere svolte in parallelo (l'input di una istruzione può dipendere dal risultato della precedente).
 - Sarebbero necessarie troppe componenti hardware per le varie unità che andrebbero poi sincronizzate.

Parallelismo a livello di processore

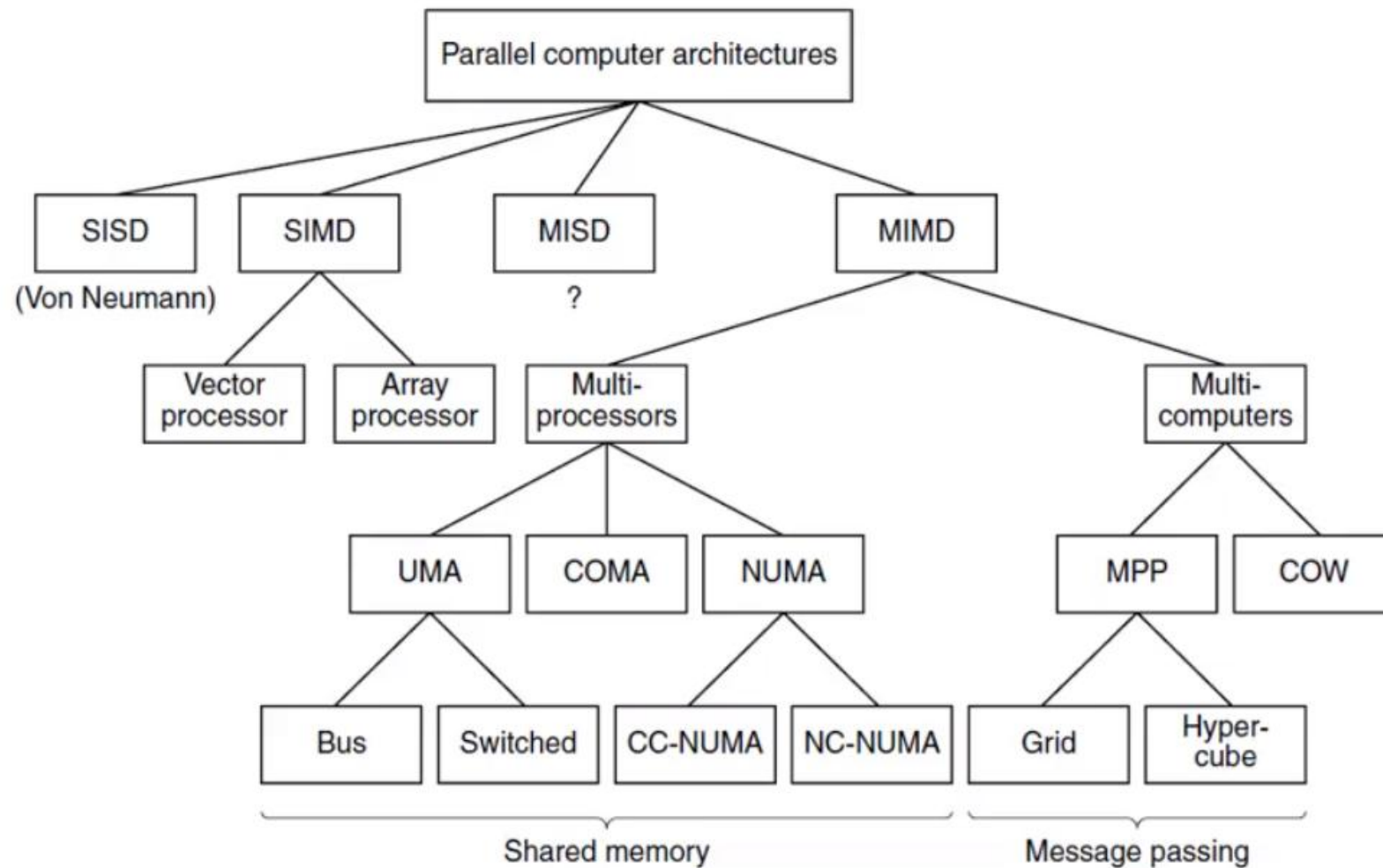
- Il parallelismo nel chip aiuta a migliorare le performance della CPU: con il pipelining e le architetture superscalari si può arrivare ad un fattore di miglioramento da **5** a **10**.
- Per incrementare drasticamente le performance di un calcolatore occorre progettare sistemi con molte CPU, in questo caso si può arrivare ad ottenere un incremento di **50**, **100**, o anche di più.
- Esistono così tre differenti approcci:
 - **Computer con parallelismo sui dati.**
 - **Multiprocessori.**
 - **Multicomputer.**

Classificazione di Flynn

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

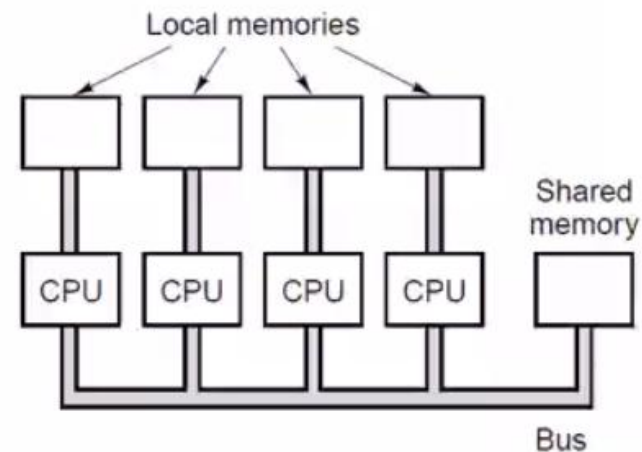
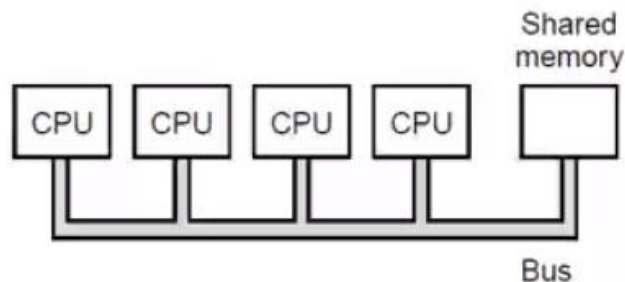
Flusso di Istruzioni	Flusso di Dati	Nome	Esempio
Singolo	Singolo	SISD	Modello di Von Neumann
Singolo	Multiplo	SIMD	Supercomputer vettoriali
Multiplo	Singolo	MISD	Non sono note
Multiplo	Multiplo	MIMD	Multiprocessori e Multicomputer

Tassonomia dei calcolatori paralleli



Multiprocessori

- È una architettura costituita da più CPU che condividono una memoria comune.
- Poiché ciascuna CPU può leggere o scrivere qualsiasi zona della memoria comune, le CPU devono sincronizzarsi via software.
- In questo caso le CPU hanno la necessità di interagire in modo così profondo che il sistema è detto fortemente accoppiato (**tightly coupled**).

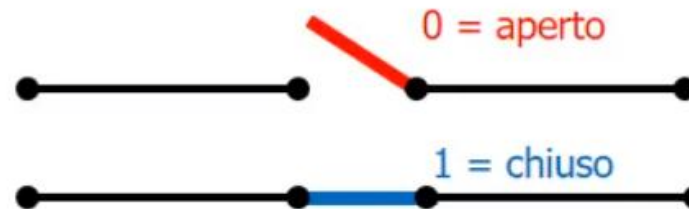


Multicomputer

- Multiprocessori con più di 256 CPU sono difficili da realizzare: si pensi ad esempio al solo problema delle connessioni di ciascuna CPU verso la memoria comune!
- I progettisti hanno superato il problema abbandonando il concetto di memoria comune e realizzando un elevato numero di CPU interconnesse, ciascuna con la propria memoria privata.
- Le CPU in un multicomputer:
 - Sono accoppiate in modo lasco (**loosely coupled**).
 - Comunicano attraverso lo scambio di messaggi.
- In architetture grandi la completa interconnessione non è fattibile così sono utilizzate topologie differenti come la griglia, l'albero o l'anello.

MEMORIA PRINCIPALE

- La memoria è quella parte del computer dove sono memorizzati programmi e dati.
- L'unità base della memoria è il **bit** (BInary digiT): una variabile che assume solo due stati (0 e 1).



- Un computer "ragiona" unicamente interpretando gruppi di bit, cioè comandi rappresentati da sequenze di "zero" e di "uno" (per esempio, 00101100).
- Convenzionalmente, 1 **byte** sono 8 bit.

I sistemi di numerazione

- I sistemi di numerazione nascono per rappresentare i numeri.
- Un sistema numerico è quindi un linguaggio per scrivere i numeri e, come tale, si compone di:
- **Alfabeto**: l'insieme di simboli.
- **Grammatica**: le regole con le quali comporre sequenze valide (**sintassi**) e quelle per dedurne il significato numerico (**semantica**).

Sistema di numerazione romano

- È un sistema di **numerazione additivo**, ovvero a ogni simbolo è associato un valore e il numero rappresentato è dato dalla somma dei valori dei simboli:
 - I = 1
 - V = 5
 - X = 10
 - L = 50
 - C = 100
 - D = 500
 - M = 1000
- Esempio: XVII (romano) = 17 (decimale)



il numero zero non si
può scrivere!



Sistema di numerazione posizionale

- Il numero di simboli utilizzati da un sistema di numerazione posizionale si chiama **base**.
- Il valore di ogni simbolo (o cifra) dipende dalla posizione che occupa nel numero, ad ogni posizione corrisponde una potenza della base.

Esempio:

$$321_{10} = 3 \cdot 10^2 + 2 \cdot 10^1 + 1 \cdot 10^0 = 3 \cdot 100 + 2 \cdot 10 + 1 \cdot 1$$

$$N_b = \sum_{j=0}^k c_j b^j$$

Sistemi di numerazione posizionali

Esempi	base	alfabeto
1011_2 10101111_2	2	$\{0,1\}$
1011_8 12375_8	8	$\{0,1,2,3,4,5,6,7\}$
1011_{10} 12375_{10}	10	$\{0,1,2,3,4,5,6,7,8,9\}$
1011_{16} 12375_H $1A3F_{16}$	16	$\{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$

Esempi di rappresentazioni numeriche

- Decimale (es. 2002_{10}):

cifre	2	0	0	2
potenze	10^3	10^2	10^1	10^0
valori	2000	0	0	2

- Binario (es. $0111_2 = 7_{10}$):

cifre	0	1	1	1
potenze	2^3	2^2	2^1	2^0
valori	0	4	2	1

Conversioni di base

- Da decimale a qualsiasi base:

Metodo delle divisioni successive

- Da qualsiasi base (b) a decimale:

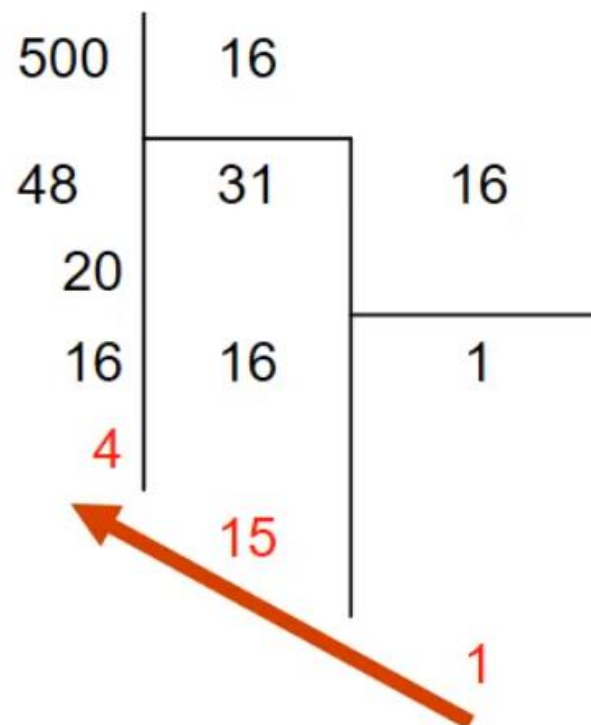
$$N_b = \sum_{j=0}^k c_j b^j$$

c_k	c_{k-1}	...	c_1	c_0
-------	-----------	-----	-------	-------

b^k	b^{k-1}	...	b^1	b^0
-------	-----------	-----	-------	-------

Conversione: base 10 \rightarrow base 16

- Trasformare 500_{10} in base 16.



Conversione: base generica \rightarrow base 10

- Trasformare $1F4_{16}$ in base 10.

cifre	1	F	4
pesi	16^2	16^1	16^0
valori	256	240	4


$$N_b = \sum_{j=0}^k c_j b^j$$

$$1F4_{16} = 1 \cdot 16^2 + F \cdot 16^1 + 4 \cdot 16^0 = 1 \cdot 256 + 15 \cdot 16 + 4 \cdot 1 = 500_{10}$$

Metodo semplificato: base 10 \rightarrow base 2

- Dividere per 2 corrisponde a stabilire se un numero è pari o dispari...

50	0
25	1
12	0
6	0
3	1
1	1
0	



$$50_{10} = 110010_2$$

Metodo semplificato: base 2 \rightarrow base 10

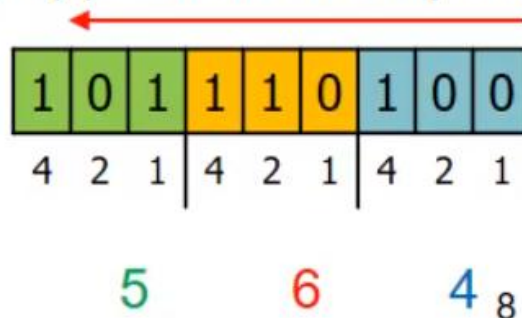
- In una somma i bit a zero possono essere esclusi, possiamo sommare le sole potenze di due dove troviamo un uno:

The diagram illustrates the conversion of the binary number 110010_2 to the decimal number 50. The binary digits are labeled with their corresponding powers of two: 5, 4, 3, 2, 1, 0. The binary number is 110010. Arrows point from the '1' bits to their respective powers of two: 32, 16, and 2. These are summed to get 50.

$$\begin{array}{r} 5 \ 4 \ 3 \ 2 \ 1 \ 0 \\ 1 \ 1 \ 0 \ 0 \ 1 \ 0_2 \\ \begin{array}{l} \rightarrow 32 \\ \rightarrow 16 \\ \rightarrow 2 \end{array} \\ \hline 50 \end{array}$$

Basi potenza l'una dell'altra

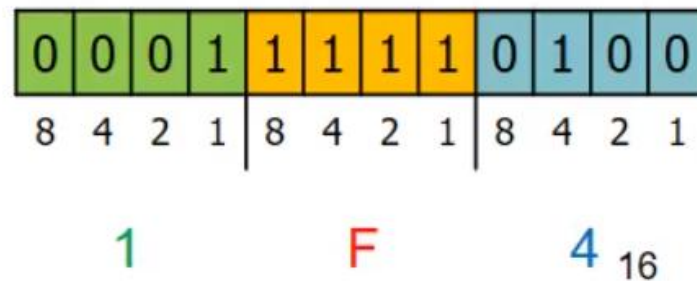
- Essendo le basi 8 e 16 potenze di 2, esiste un procedimento semplice di trasformazione dall'una all'altra base:
- Base 2 \rightarrow base 8: si suddivide il numero binario in gruppi di 3 bit (a partire dalla cifra meno significativa) e si trasforma ogni gruppo nella corrispondente cifra ottale:



- Base 8 \rightarrow base 2: ad ogni cifra ottale corrispondono 3 bit, si usano le potenze di due (4 2 1) per comporre la cifra in ottale.

base 2 (2^1) \leftrightarrow base 16 (2^3)

- Base 2 \rightarrow base 16: si suddivide il numero binario in gruppi di 4 bit (a partire dalla cifra meno significativa) e si trasforma ogni gruppo nella corrispondente cifra esadecimale:



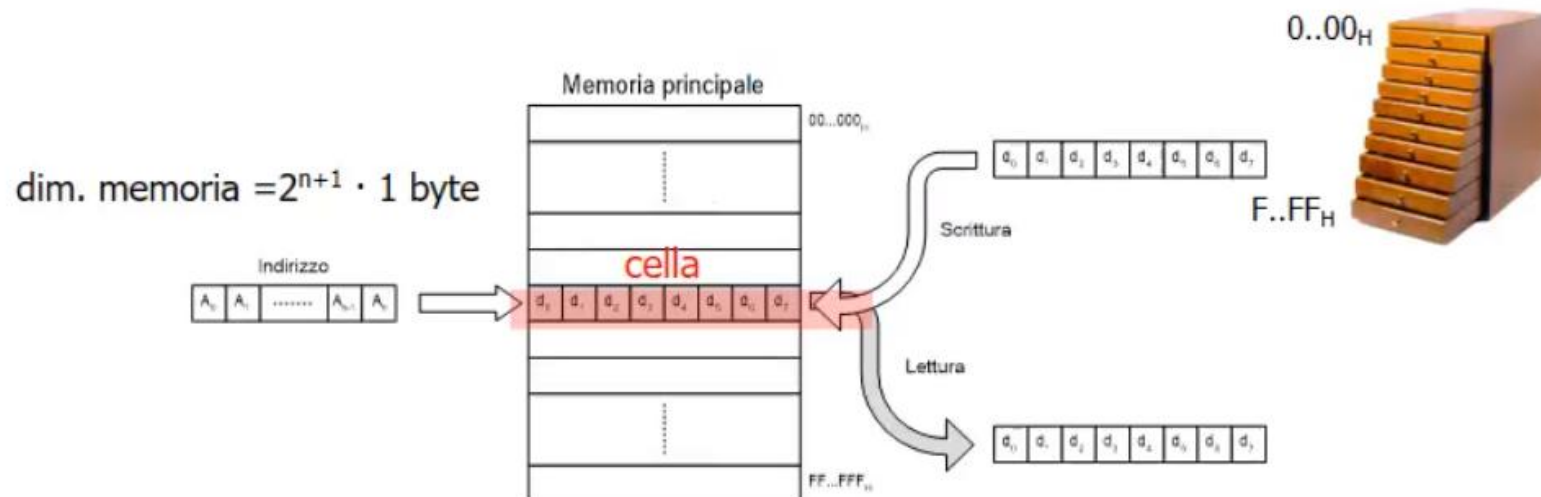
- Base 16 \rightarrow base 2: ad ogni cifra esadecimale corrispondono 4 bit, si usano le potenze di due (8 4 2 1) per comporre la cifra in esadecimale.

Confronto tra numeri

Decimale	Binario	Ottale	Esadecimale
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13

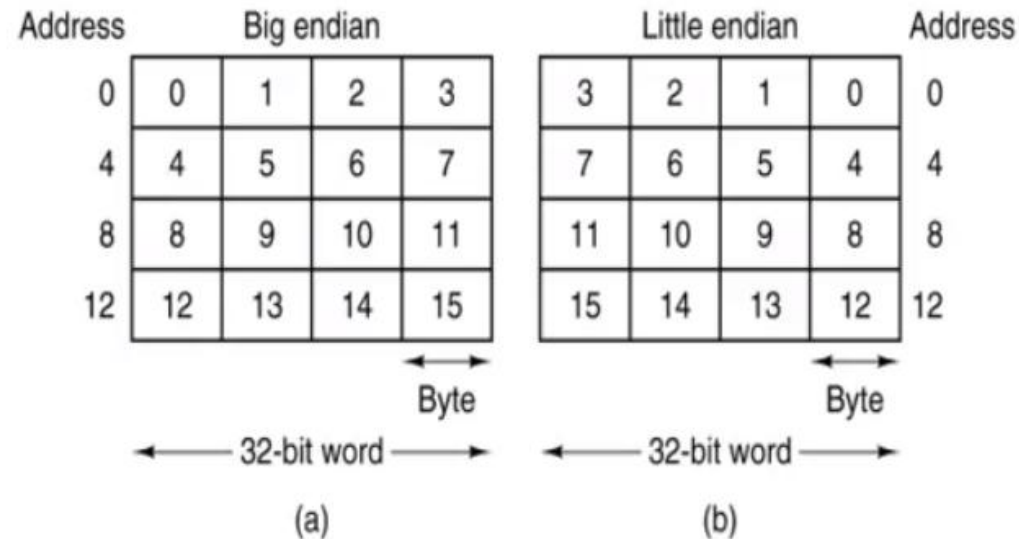
Indirizzi di memoria

- La memoria è organizzata in **celle** (o **locazioni**) identificabili da una posizione (l'**indirizzo** della cella).
- In ogni cella c'è un lo stesso quantitativo di informazione binaria (espresso in byte).
- La cella rappresenta l'unità più piccola indirizzabile.
- I byte possono essere raggruppati in **parole** (o **words**).



Ordinamento dei byte

- I bytes in una parola possono essere scritti da sinistra a destra (**Big endian**) oppure da destra a sinistra (**Little endian**).



- Esempio: il numero $0A1F_H$ può essere scritto come sequenza big endian $F1A0$ oppure come sequenza little endian così come è scritto.

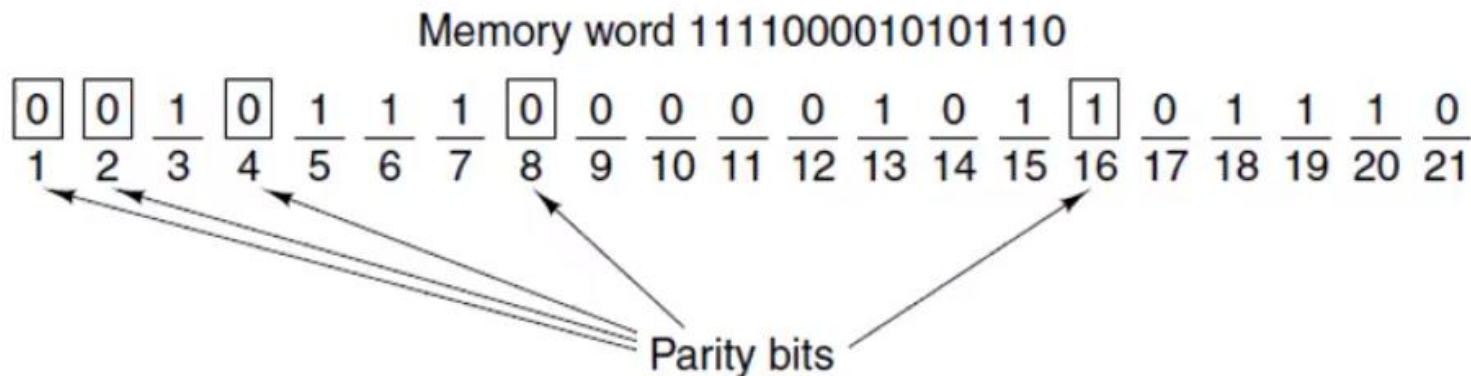
Codici di correzione di errore

- I computer possono, occasionalmente, commettere degli errori: errori, al fine di rilevare e correggere l'eventuale errore si possono aggiungere dei bit extra alla 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. Per il calcolo è sufficiente sommare il risultato dell'EXOR bit-a-bit delle due parole.
- La percentuale di overhead decrementa al crescere della dimensione della parola.

Word size	Check bits	Total size	Percent 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

Codice di Hamming

- Il bit 1 controlla i bit 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21.
- Il bit 2 controlla i bit 2, 3, 6, 7, 10, 11, 14, 15, 18, 19.
- Il bit 4 controlla i bit 4, 5, 6, 7, 12, 13, 14, 15, 20, 21.
- Il bit 8 controlla i bit 8, 9, 10, 11, 12, 13, 14, 15.
- Il bit 16 controlla i bits 16, 17, 18, 19, 20, 21.

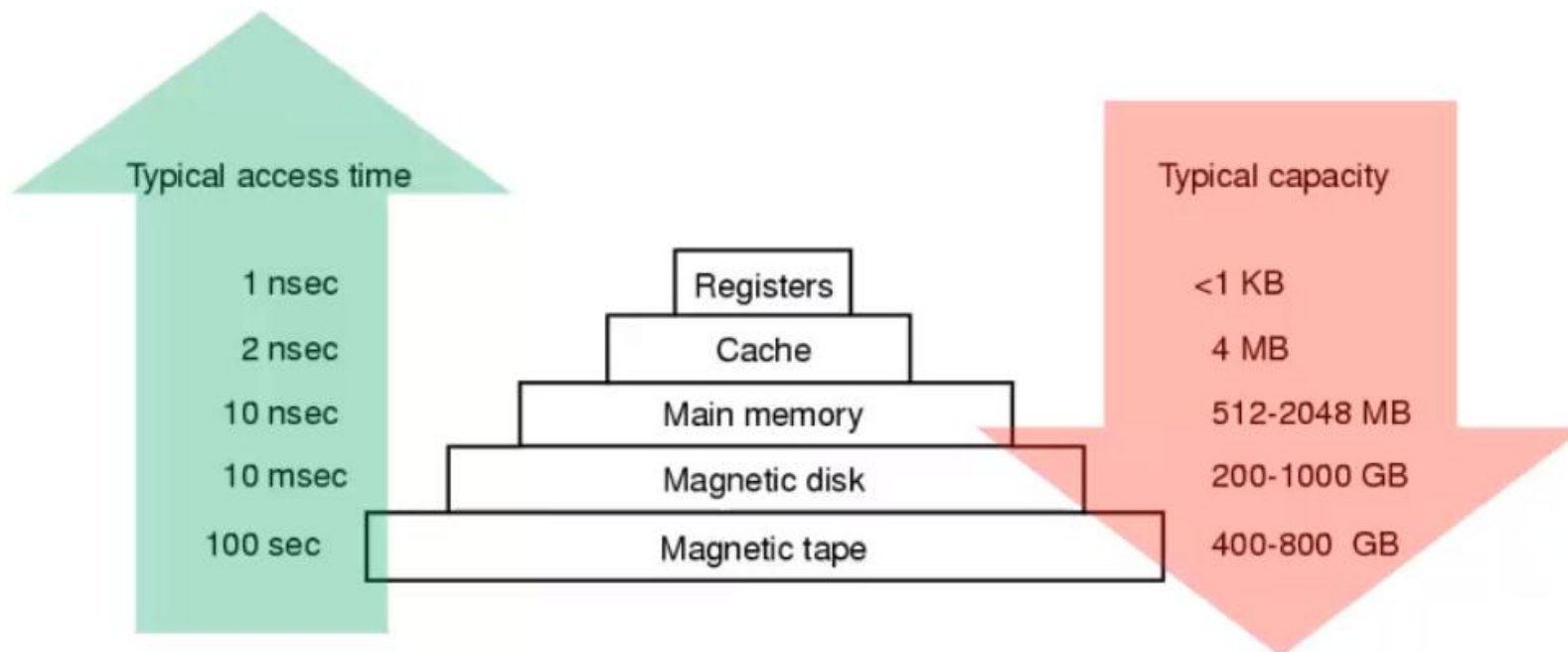


- La percentuale di overhead decrementa al crescere della dimensione della parola.

Memorie cache

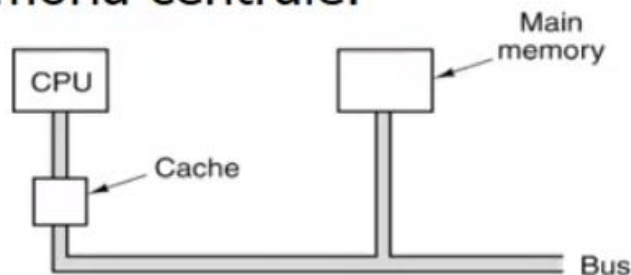
- CPU sono più veloci delle memorie.
- I progettisti di memorie incrementano la capacità, non la velocità.
- La CPU deve così aspettare molti cicli macchina prima di ricevere i dati dalla memoria.
- Gli ingegneri non riescono ad inserire nei chip delle CPU memorie veloci (e grandi) poiché questo incrementerebbe la dimensione ed il prezzo!

Gerarchie di memorie



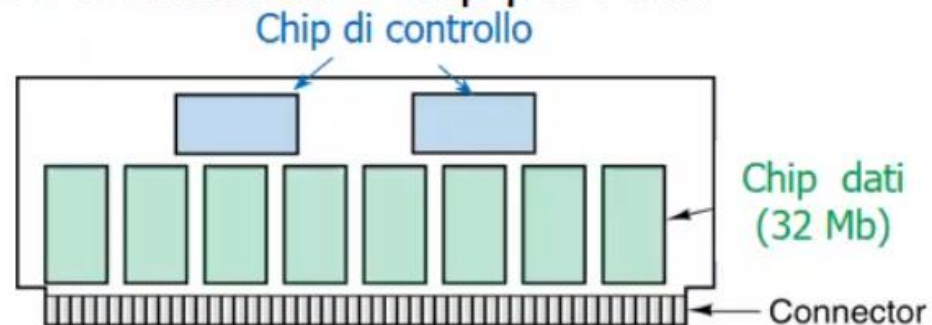
Memorie cache

- 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 piccoli 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 prima nella cache (più veloce) e poi nella memoria centrale.



Assemblaggio e tipi di memoria

- Un gruppo di integrati, tipicamente 8 o 16, è montato su un piccolo circuito stampato e venduto come unità.
- Questa unità è chiamata **SIMM** (Single Inline Memory Module) quando ha una riga di connettori su un solo lato della scheda, oppure **DIMM** (Dual Inline Memory Module) quando ha due righe di connettori su entrambi i lati della scheda.
- Le SIMM riescono a trasferire 32 bits per ciclo di clock mentre le DIMM il doppio della banda (64 bits/ciclo di clock).
- Una SIMM da 256 Mbyte è formata da 8 chip per i dati ciascuno di 32 Mbyte e due circuiti integrati destinati al controllo della scheda.



12	11	10	9	8	7	6	5	4	3	2	1
				b_3				b_2		b_1	b_0

Il testo dell'esercizio richiede di trasmettere la sequenza binaria che corrisponde al numero $F5_H$ cioè 11110101_2 . Per calcolare i bit di controllo si inseriscono in ordine i bit della parola:

12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	b_3	0	1	0	b_2	1	b_1	b_0
	■		■		■		■		■		□
	■	■			■	■			■	□	
■					■	■	■	□			
■	■	■	■	□							

 b_0 b_1 b_2 b_3

Quindi si calcola l'exor dei bit nelle posizioni indicate:

$$b_0 = 1 \oplus 1 \oplus 0 \oplus 0 \oplus 1 = 1$$

$$b_1 = 1 \oplus 1 \oplus 0 \oplus 1 \oplus 1 = 0$$

$$b_2 = 1 \oplus 0 \oplus 1 \oplus 0 = 0$$

$$b_3 = 1 \oplus 1 \oplus 1 \oplus 1 = 0$$

La sequenza da t

teams.microsoft.com sta condividendo il tuo schermo.

Interrompi condivisione

Nascondi

2.1.3 Codice di Hamming

Esercizio n.13 Supponendo di voler trasmettere la sequenza binaria corrispondente al numero $F5_H$, individuare i bit di controllo da aggiungere alla parola se si utilizza il codice di Hamming. Dimostrare con degli esempi il buon funzionamento del sistema di correzione quando si verificano degli errori sia sui bit dati e sia su quelli di controllo.

Svolgimento

Con riferimento al codice di correzione di Hamming, si ricorda che per trasferire una parola dati di m bit e n bit di controllo deve valere la relazione $m + n + 1 \leq 2^n$. Quindi essendo la parola da trasferire di $m = 8$ bit, sono necessari 4 bit di controllo (b_3, b_2, b_1, b_0) per un totale di 12 bit trasferiti. In particolare il bit b_i controlla i bit che si trovano nelle posizioni che contengono il peso 2^i , così come indicato:

bit	peso	posizioni controllate
b_0	2^0	$\square\square\square 1_2$
b_1	2^1	$\square\square 1 \square_2$
b_2	2^2	$\square 1 \square \square_2$
b_3	2^3	$1 \square \square \square_2$

Quindi occorre inserire i valori della parola da trasferire nel seguente schema:

teams.microsoft.com sta condividendo il tuo schermo.

Interrompi condivisione

Nascondi

