

Corso di Architettura degli Elaboratori

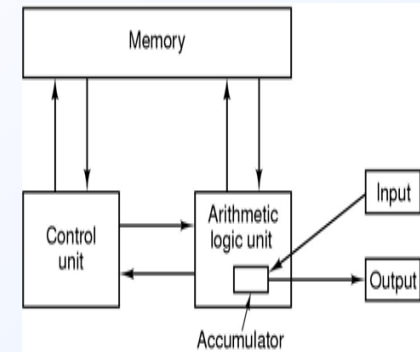
Struttura dei Calcolatori: CPU e Memoria Principale

Matteo Baldoni

Dipartimento di Informatica
Università degli Studi di Torino
C.so Svizzera, 185 – I-10149 Torino
baldoni@di.unito.it
<http://www.di.unito.it/~baldoni>

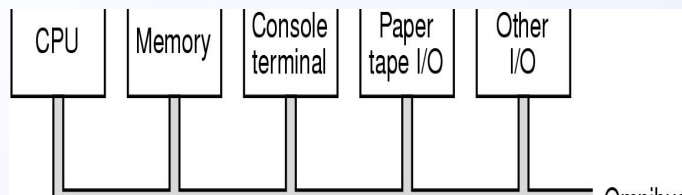
Macchina di von Neumann

- × Von Neumann collaborò prima all'ENIAC (lenta aritmetica decimale), quindi alla macchina IAS (aritmetica binaria)
- × Cinque parti principali: memoria, ALU, unità di controllo e dispositivi di I/O
- × Registro accumulatore
- × È alla base di quasi tutti i calcolatori digitali (dal 1949!)



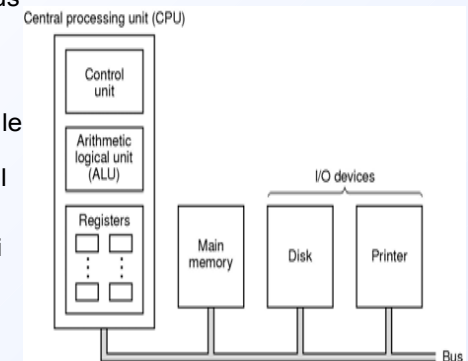
PDP-8: introduzione del bus

- × 1995-1965: seconda generazione, transistor
- × PDP-8 della DEC introduce per la prima volta un bus singolo, un insieme di fili usato per collegare i componenti di un calcolatore



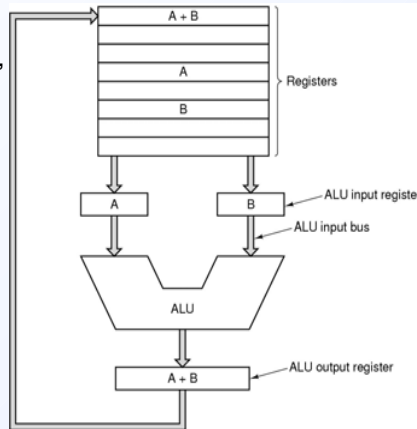
Struttura di un calcolatore

- × Struttura di un semplice calcolatore orientato a bus (collegati mediante un bus, sia interni che esterni)
- × Unità di controllo: legge le istruzioni della memoria centrale e ne determina il tipo
- × ALU: esegue le istruzioni
- × Registri: memoria temporanea ad alta velocità (Program Counter, PC, Instruction Register, IR)



Central Processing Unit

- × **Data Path:** organizzazione interna di una CPU (registri, ALU, bus interno)
- × Registro: memorie veloci per dati temporanei
- × Registri A e B, input dell'ALU, e A+B output dell'ALU
- × Istruzioni registro-registro e registro-memoria
- × **Ciclo del data path**



Esecuzione delle istruzioni

1. Prendi l'istruzione seguente dalla memoria e mettila nel registro delle istruzioni
2. Cambia il program counter per indicare l'istruzione seguente
3. Determina il tipo dell'istruzione appena letta
4. Se l'istruzione usa una parola in memoria, determina dove si trova
5. Metti la parola, se necessario, in un registro della CPU
6. Esegui l'istruzione
7. Torna al punto 1 e inizia a eseguire l'istruzione successiva

Un interprete per l'esecuzione

- × Interprete: un programma per eseguire le istruzioni di un altro programma
- × Maurice Wilkes (1951) introduce la **microprogrammazione**: correzione errori, estensione, test
- × **Architettura del System/360 IBM**
- × **Control store**

```
public class Interp {
    static int PC, AC;
    static int instr, instr_type;
    static int data_loc, data;
    static boolean run_bit = true;

    public static void interpret(int memory[],
        int starting_address) {
        PC = starting_address;
        while (run_bit) {
            instr = memory[PC];
            PC = PC + 1;
            instr_type = get_instr_type(instr);
            data_loc = find_data(instr, instr_type);
            if (data_loc >= 0)
                data = memory[data_loc];
            execute(instr_type, data);
        }
    }
}
```

Un interprete per l'esecuzione

- × Motorola 68000 (successore del 6800) vs Zilog Z8000 (successore dello Z80)
- × VAX della DEC (Digital Equipment Corporation), fine anni '70: centinaia di istruzioni e più di 200 modi diversi di specificare gli operandi
- × Minore distanza tra quello che poteva fare la macchina e i linguaggi di "alto livello"
- × Vantaggi:
 - ✓ Correzione dell'implementazione delle istruzioni anche presso il cliente
 - ✓ Opportunità di aggiungere nuove istruzioni
 - ✓ Migliore progettazione strutturata

RISC vs CISC

- × David Patterson e Carlo Séquin, Berkeley 1980, coniano il termine **RISC** (*Reduced Instruction Set Computer*) come concetto guida della loro CPURISC I e RISC II che **non usava l'interpretazione** (poi diventata SPARC)
- × John Hennessy, Stanford 1981, progetta e costruisce MIPS
- × Si contrappongono al VAX, Intel, IBM, **CISC** (*Complex Instruction Set Computer*)
- × Numero limitato di istruzioni eseguite velocemente (un solo ciclo del data path)
- × 1 istruzione CISC = 5/6 istruzione RISC ma 1 istruzione RISC 10 volte piu` veloce di 1 istruzione CISC
- × Nota: anni '80 velocita` memoria centrale ~ velocita` control store

RISC vs CISC

- × Vera novita` introdotta con i RISC:
 - ✓ *e` importante progettare istruzioni che possano essere messe in esecuzione velocemente*
 - ✓ *non importa quanto tempo un'istruzione abbia bisogno per essere completata ma e` importante che se ne possano iniziare il piu` possibile al secondo*

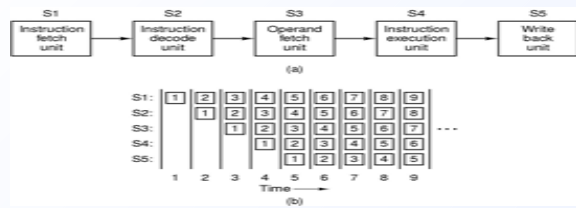
RISC vs CISC

- × Intel con il 486 applica la stessa filosofia dei RISC anche all'architettura CISC
- × Ogni CPU 486 (e successive) contengono un piccolo processore RISC che esegue le istruzioni piu` semplici (e piu` comuni) in un ciclo di data path, le istruzioni piu` complesse vengono interpretate
- × Non veloce come un RISC puro ma compatibile (soprattutto a livello di software) con i modelli precedenti (estremamente importante per il mercato)
- × Non c'e` un vincitore ma si e` appreso molto sulla "buona progettazione" di un calcolatore

Principi di progettazione RISC

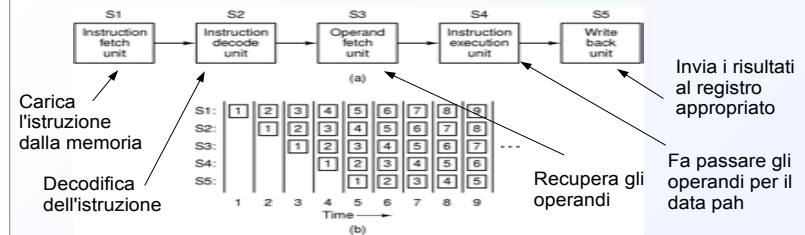
- × E` importante tenere sempre conto dei cambiamenti tecnologici (vedi velocita` della memoria centrale e del control store)
- × Tutte le istruzioni vengono eseguite direttamente dall'hardware (nota: le istruzioni complesse e meno frequenti dei CISC vengono eseguite da piu` microistruzioni)
- × Ottimizzare la velocita` con la quale vengono iniziate le istruzioni: il parallelismo gioca un ruolo fondamentale
- × Le istruzioni dovrebbero essere facilmente decodificabili: istruzioni con una struttura regolare
- × Solo le istruzioni di load e store dovrebbero contenere indirizzi di memoria: e` piu` facile sovrapporre ad altre istruzioni
- × Disporre di molti registri (almeno 32): per accedere meno alla memoria

Pipelining



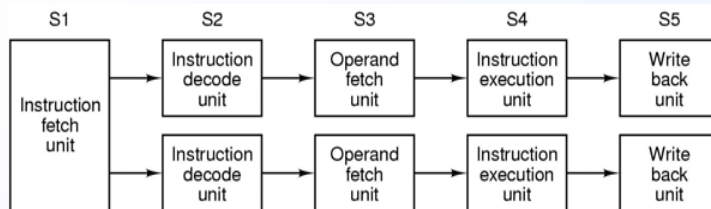
- × Stretch IBM (1959) introduce il registro *prefetch buffer* (due fasi di esecuzione: lettura e esecuzione propria)
- × **Parallelismo a livello delle istruzioni**
- × Pipelining: suddividere l'esecuzione in molte fasi, ognuna gestita da un pezzo di hardware dedicato
- × Ogni componente hardware puo` eseguire in parallelo alle altre componenti il proprio compito

Pipelining



- × Ogni istruzione passa attraverso una serie di fasi di lavorazione prima di essere completata
- × Esempio: ciclo macchina 2 nsec, quindi 10 nsec per istruzione, la macchina funziona a 500 MIPS (*Millions of Instructions Per Second*) anziche` 100 MIPS
- × Le pipeline sono introdotte dalle macchine RISC

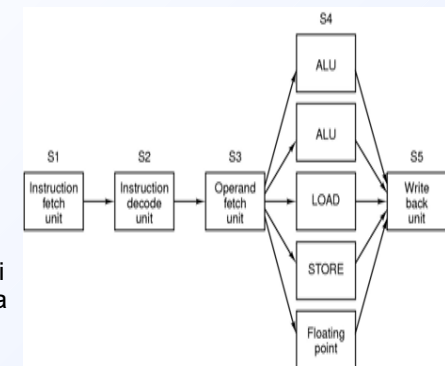
Architetture superscalari



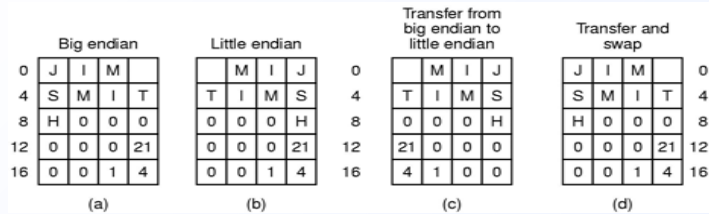
- × Due e` meglio di uno
- × La Intel introduce la singola pipeline con il 486 e la pipeline doppia con il Pentium (dette pipeline u e v):
 - ✓ la pipeline u esegue qualunque istruzione
 - ✓ la pipeline v solo istruzioni semplici
- × Il compilatore deve garantire il corretto uso

Architetture superscalari

- × Introdurre piu` pipeline e` complesso
- × Pipeline singola con unita` funzionali multiple
- × Pentium II (ma introdotta per la prima volta nel CDC 6600 30 anni prima)
- × Idea di base: lo stadio S3 puo` inviare le istruzioni molto piu` velocemente di quanto lo stadio S4 possa eseguirle



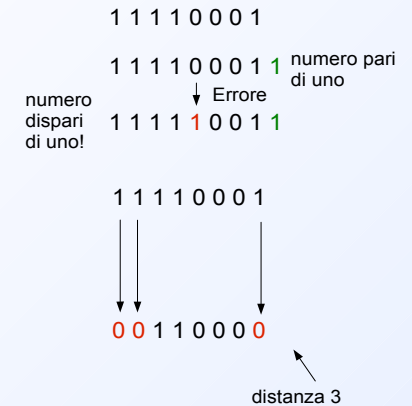
Ordinamento dei byte



- × Problemi per la trasmissione byte per byte di informazioni di tipo alfanumeriche e numeriche: inversione del numero (c), oppure delle stringhe (d)
- × E' necessario includere l'indicazione di che tipo di dato si tratta (stringa, numero intero, altro)!
- × Mancanza di uno standard

Codici di correzione degli errori

- × Aggiungere dei bit a ogni parola della memoria in modo da identificare e correggere eventuali errori
- × n bit = m bit per la parola + r bit di ridondanza (o bit di controllo)
- × **Distanza di Hamming**: se due parole sono distanti d , ci vorranno d errori di un bit per convertire una nell'altra



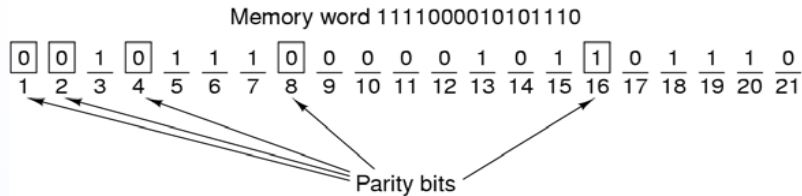
Codici di correzione degli errori

- × Parola di m bit: solo 2^m su 2^n combinazioni sono valide
- × Dato l'algoritmo per determinare i bit di controllo e' possibile stabilire se una parola di n bit e' valida oppure no
- × Se la parola non e' valida il calcolatore puo' segnalare l'occorrenza di un errore
- × La minima distanza tra due parole del codice e' la distanza di Hamming per tutto il codice (il semplice bit di parita' visto prima ha distanza 2)
- × Per individuare d errori di un bit serve un codice con distanza almeno $d + 1$, per correggere d errori serve un codice con distanza almeno $2d + 1$

Codici di correzione degli errori

- × **Esempio**: si desidera costruire un codice per correggere tutti gli errori di un bit
- × $n = m + r$
 - ✓ ognuna delle 2^m parole ha n parole di codice illegali a distanza 1 (si inverte ogni bit)
 - ✓ ognuna delle 2^m parole richiede $n + 1$ configurazioni (1 valida e n no valide)
 - ✓ il numero totale delle configurazioni deve soddisfare la disequazione: $(n + 1)2^m \leq 2^n$, cioe' $(m + r + 1)2^m \leq 2^n$
 - ✓ per $m = 8, 16, 32, n = 12, 21, 38$, cioe' $r = 4, 5, 6$

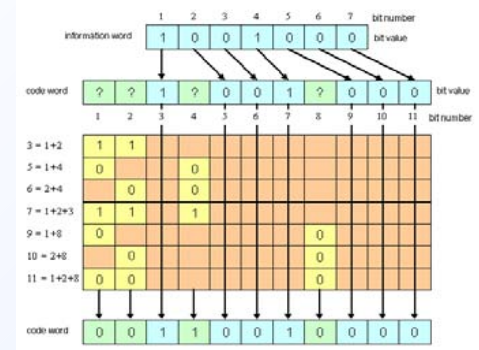
Algoritmo di Hamming



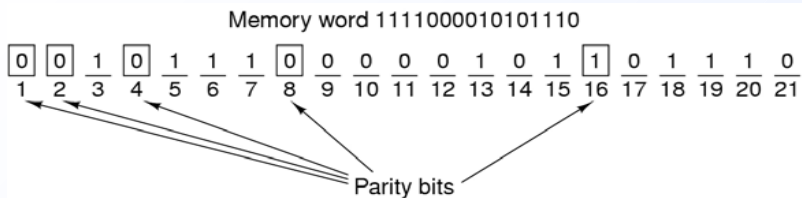
- × I bit sono numerati da 1 in avanti
- × I bit il cui numero di bit e' una potenza di due e' un bit di controllo (o anche di parita'), i rimanenti per i dati
- × Ogni bit controlla specifiche posizioni di bit: il bit b -esimo e controllato dai bit di parita' di posizione b_1, b_2, \dots, b_j tale che $b = b_1 + b_2 + \dots + b_j$
- × Esempio, parita' pari: il bit di parita' controlla che il numero totale di 1 nelle posizioni controllate sia pari

Algoritmo di Hamming

- × Esempio: $m = 7$, quindi $n = 11$ e $r = 4$
- × Ogni bit controlla specifiche posizioni di bit: il bit b -esimo e controllato dai bit di parita' di posizione b_1, b_2, \dots, b_j tale che $b = b_1 + b_2 + \dots + b_j$
- × Parita' pari: il bit di parita' controlla che il numero totale di 1 nelle posizioni controllate sia pari

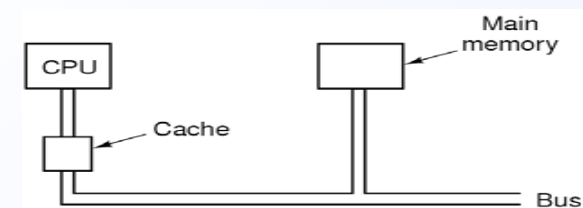


Algoritmo di Hamming



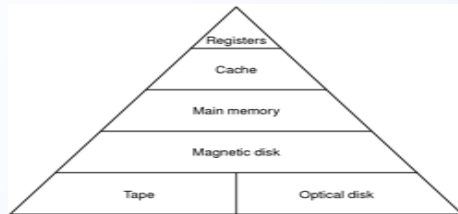
- × I bit di parita' vengono controllati, se vi e' un errore:
- × Sommare le posizioni di tutti i bit di parita' non corretti, tale valore corrisponde alla posizione del bit non corretto (e va invertito)
- × Nota: se sono presenti piu' di un errore il metodo non funziona (distanza di Hamming 3)

Memoria cache



- × Un problema di economia!
- × L'uso della memoria di cache e' basata sul **principio di localita'** del codice: i programmi non accedono alla memoria a caso
- × Le parole di memoria piu' usate vengono tenute in una cache

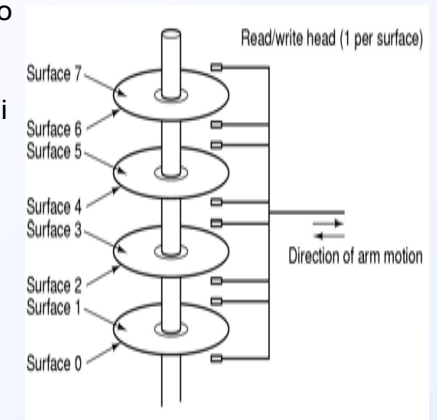
Gerarchia della memoria



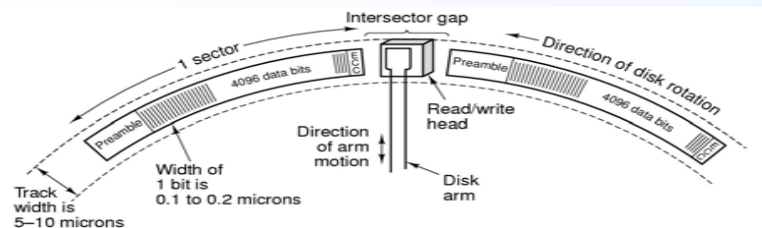
- x Gerarchia della memoria secondo la capacità di memorizzazione
- x Questa gerarchia rispetta anche i costi di memorizzazione (più in basso i più economici, più in alto i più costosi)
- x Rispetta anche i tempi di accesso all'informazione

Dischi magnetici

- x Alluminio con rivestimento magnetizzabile
- x Accesso dell'ordine dei millisecondi (nanosecondi per i registri!)
- x Sigillati in fabbrica
- x Controllore del disco
- x Organizzazione a cilindri (tracce alla stessa distanza dal centro)
- x Tempo di seek
- x Latenza di rotazione



Dischi magnetici



- x Ogni traccia è organizzata in settori
- x Preambolo per la sincronizzazione della testina
- x Insieme dei dati
- x Codice correzione degli errori (Hamming o Reed-Salomon)
- x Gap tra settori

Dischi magnetici

- x Dischi Winchester
- x Floppy disk
- x Dischi IDE (Integrated Drive Electronics) anni '80 e EIDE (Extended IDE)
- x Dischi SCSI (Small Computer System Interface), 1986, e SCSI-2, 1994
- x RAID (Redundant Array of Inexpensive/Independent Disk) e striping