Appunti di Architettura degli Elaboratori

Andreas Araya Osorio

January 21, 2022

Contents

Chapter 1

Prima Parte

1.1 Rappresentazione dei numeri

I numeri possono essere rappresentati in qualsiasi base, noi utilizziamo la base 10 come conseguenza del numero delle nostre dita.
I **calcolatori** utilizzano la base 2, ovvero il binario, che può facilmente essere ricondotta alla condizione di uno stato elettrico o **positivo (1)** o **negativo (0)**
Per la rappresentazione di un numero in qualsiasi base:

Definizione 1:
La sequenza di k cifre:

$$d_k \cdot d_{k-1} \cdot \cdots \cdot d_1 \cdot d_0 \tag{1.1}$$

E questa sequenza la moltiplichiamo per la base scelta:

$$d_k \times b^k \cdot d_{k-1} \times b^{k-1} \cdot \cdots \cdot d_1 \times b^1 \cdot d_0 \times b^0 \tag{1.2}$$

*dove **b** è la base da noi scelta.*

Definizione 2 (Numero di cifre in una base N)**:**
In una qualsiasi base N il numero di cifre equivale a:

$$cifre = N-1, N-2, \ldots, 1, 0. \tag{1.3}$$

1.1.1 Notazione

Definizione 3 (BIT)**:**
***Bit** = binary digit, uno dei due simboli (0, 1) del sistema numerico binario. Esso è l'unità elementare dell'informazione trattata da un elaboratore.*
Numeri di 8, 16, 32 bit equivale in base 10 a parlare di numeri a 3, 4, 5, ... cifre

Definizione 4 (BYTE)**:**
***Byte** = 8 bit, è una sequenza di bit, convenzionalmente l'unità di misura delle capacità di una memoria.*
Può assumere $2^8 = 256(0 - 255)$ *possibili valori*

Definizione 5 (Parola)**:**
Word/Parola *= corrisponde a 16, 32 o 64 bit in base al tipo di IS (Instruction Set), essa è l'unità più piccola di informazione su cui un elaboratore può intervenire.*

Nome	Simbolo	Multiplo in base 10	Multiplo in base 2
chilobyte	kB	10^3	2^{10}
megabyte	MB	10^6	2^{20}
gigabyte	GB	10^9	2^{30}
terabyte	TB	10^{12}	2^{40}
petabyte	PB	10^{15}	2^{50}

Convenzionalmente si utlizzano come unità di misura:

- il B(yte) per la capacità di una memoria
- il b(it)/s per la velocità di trasmissione di dati.

1.1.2 Binario

Con la base 2 abbiamo le cifre dallo 0 allo 1.
Per la definizione precedente la rappresentazione sarà:

$$d_k \times 2^k \cdot d_{k-1} \times 2^{k-1} \cdot \cdots \cdot d_1 \times 2^1 \cdot d_0 \times 2^0 \tag{1.4}$$

Definizione 6 (Valore minimo e massimo)**:**
Il valore minmo e massimo di un numero di n cifre è:

- ***Valore minimo*** $= 000000 \ldots 00(n\ times) = 0$
- ***Valore massimo*** $=111111 \ldots 11(n\ times) = 2^n - 1$

$$2^{n-1} + 2^{n-2} + \cdots + 2^2 + 2^1 + 2^0 = 2^n - 1 \tag{1.5}$$

Esempio 1.

$$n = 3 \implies 111 = 2^2 + 2^1 + 2^0 = 7 = 2^3 - 1 = 8 - 1 \tag{1.6}$$

1.1.3 Decimale

Nella rappresentazione decimale abbiamo le cifre dallo 0 al 9.
Un qualsiasi numero in base decimale si può rappresentare come:

$$d_k \times 10^k \cdot d_{k-1} \times 10^{k-1} \cdot \cdots \cdot d_1 \times 10^1 \cdot d_0 \times 10^0 \tag{1.7}$$

Binario	Esadecimale
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9

1.1.4 Ottale

Nella rappresentazione ottale si hanno le cifre dallo 0 al 7.
Questa base viene utilizzata perchè essendo un multiplo di 2 si possono facilmente convertire numeri ottali in binario:
Con 3 cifre si possono rappresentare 8 bit. In questo modo si possono avere numeri più facilmente maneggiabili da umani, rispetto a lunge stringhe di 0 o 1.

Binario	Ottale
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7

1.1.5 Esadecimale

Nella rappresentazione esadecimale si hanno 15 cifre: dallo 0, ..., 9, A, ..., F.
Un simbolo (cifra) in questa base rappresenta 4 cifre binarie (4 bit).
Con 4 cifre esadecimali si possono rappresentare 16 bit.

Binario	Esadecimale
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

1.1.6 Algebra di Boole

Essa viene utillizzata per la specifica di funzioni logiche.
Una qualsiasi variabile può assumere 2 valori: **vero** o **falso**

Definizione 7 (Operazoini logiche di base)**:**

$$\begin{aligned} A\ AND\ B &= A \cdot B \\ A\ OR\ B &= A + B \\ NOT\ A &= \overline{A} \end{aligned} \tag{1.8}$$

A	B	$A\ AND\ B$	$A\ OR\ B$	$NOT\ A$
0	*0*	*0*	*0*	*1*
0	*1*	*0*	*1*	*1*
1	*0*	*0*	*1*	*0*
1	*1*	*1*	*1*	*0*

Si indica con: $\oplus = xor$, è l'or esclusivo, esso è vero solo quando una delle due variabili è vera, non quando lo sono entrambe.
$\overline{A \cdot B} = nand$, l'opposto dell'AND classico
Grazie a XOR e NAND si possono rappresentare tutte le altre funzioni logiche attraverso delle combinazioni di questi due.

A	B	$(\overline{A})$	A · B	A + B	$\overline{A \cdot B}$	$\overline{A + B}$	A ⊕ B
0	0	1	0	0	1	1	0
0	1	1	0	1	1	0	1
1	0	0	0	1	1	0	1
1	1	0	1	1	0	0	0

1.2 APPROACH

There's a difference between
Computer architecture and **Computer organization**.

- C.Architecture = attributes of a system visible to a programmer.
  ISA = Instruction set architecture, is a synonym of C.A.
- C.Organization = operational units and their interconnections that realize the architectural specifications. Examples:
  - instruction set
  - number of bits used to represent various data types
  - I/O mechanisms and techniques for adressing memory

1.2.1 Structure and function

A modern computer is a hierarchical system, and is a set of interrelated subsytems. These have, in turn, subsystems of their own until we reach the lowest level of subsystems. There's a huge difference between:

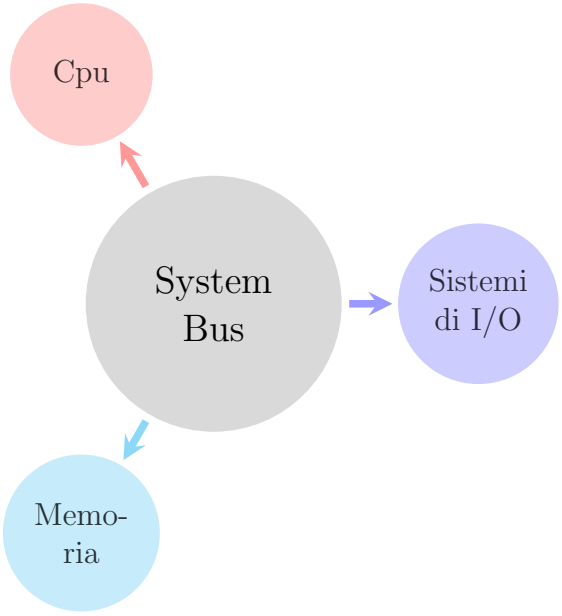
- **structure** the way components are interrelated
- **function** the operation of each individual

The way that is considered most efficient and the clearest approach for describing pc components is:**TOP-DOWN**

1.3 Componenti principali

Le componenti principali di un elaboratore sono:

- CPU
- Memoria
- Sistemi di I/O
- Interconnessioni (system bus)



Definizione 8 (Architettura di Von Neumann)**:**
Secondo questo tipo di architettura, un elaboratore è composto di questi principali componenti:

- *dati e istruzioni in memoria*
- *memoria accessibile per indirizzo*
- *esecuzione sequenziale delle istruzioni*

Definizione 9 (Programma cablato)**:**
Consiste nel costruire i componenti logici in modo tale che il risultato sia quello voluto e non può essere modificato in seguito.
Vuol dire "programmare" a livello hardware, ovvero con le componenti fisiche.
Non è un sistema flessibile, esegue solo operazioni predeterminate.

Definizione 10 (Programma)**:**
Un programma è una sequenza di passi
Ogni passo corrsiponde ad un'operazionie logica.
Ogni operazione determina un diverso insieme di segnali di controllo.

Definizione 11 (Programmazione software)**:**
Nasce con Von Neumann, si parte da un hardware generico, si ha una parte che preleva il codice di una istruzione, è generale: L'hardware di cui parliamo si dice ***general purpose****, utile a vari scopi. Si hanno poi dei segnali di controllo corrispondenti.*
Questo sistema è molto più flessibile di quello "cablato".
La CPU assume delle funzioni diverse ovvero:

- *interprete delle istruzioni*
- *generico modulo per operazioni aritmetico logiche = ALU*

I segnali di controllo sono necessari per far eseguire al giusto modulo la giusta operazione: ALU
ALU prende segnali di controllo ed esegue le istruzioni codificate
In questo sistema si ha la codifica delle istruzioni e la decodifica delle istruzioni.

Definizione 12 (Memoria principale nell'architettura di Von Neumann)**:**
Si ha la possibilità di salti oltre che all'esecuzione sequenziale(in serie)
Per esempio con le operazioni che richiedono accesso a più dati in memoria nello stesso momento.
Inoltre essa ha il compito di immaggazzinare dati e istruzioni

Esempio 2.
Somma con 2 numeri in locazione di memoria diverse

1.4 CPU

Essa non deve solo eseguire istruzioni ma anche gestire dei segnali di controllo e getstire delle risorse.
Composta da Vari componenti principali:

- **EU** = execution unit = alu
- **IR** = instruction register, registro che contiene l'istruziuone da eseguire successivamente a quella nel PC.
- **PC** = program counter, puntatore all'istruzione indirizzo dell'istruzione da eseguire presente nella memoria.
- **MAR** = memory address register, registro di interfaccia con il bus di sistema, contiene solo registri
- **MBR** = memory buffer register, contiene solo dati.
  Il MAR e il MBR mantengono le infromazioni fino a che non è disponibile il bus di sistema per essere impiegato.
  - in caso di lettura raccolgono il dato dal bus.
  - in caso di scrittura contengono il dato.
- **I/O AR** indirizzo periferica con cui scambiare dati, specificare periferica
- **I/O BR** raccolta dati

Quando si ha un salto nell'esecuzione delle istruzioni incrementa l'indirizzo del PC
Inoltre è presente un buffer nel modulo I/O: è una memoria interna al sistema di input output, esso serve perchè la CPU invia dati troppo velocemente rispetto ed esso non può riceverli alla stessa velocità, per questo il buffer dell'I/O, mantiene in memoria i dati inviati dalla più veloce CPU

1.4.1 Funzionamento CPU

- **Fetch**: reperimento, prelievo dell'istruzione dalla memoria
- **Execute**: esecuzione dell'istruzione prelevata dalla memoria

Il registro **PC** contiene l'indirizzo di memoria della cella di Memoria contenente l'istruzione da eseguire. Quando si ha un prelievo di istruzioni dalla memoria, si ha un incremento del PC. L'istruzione prelevata viene messa in IR poi viene eseguita.

1.4.2 Tipi di Operazioni

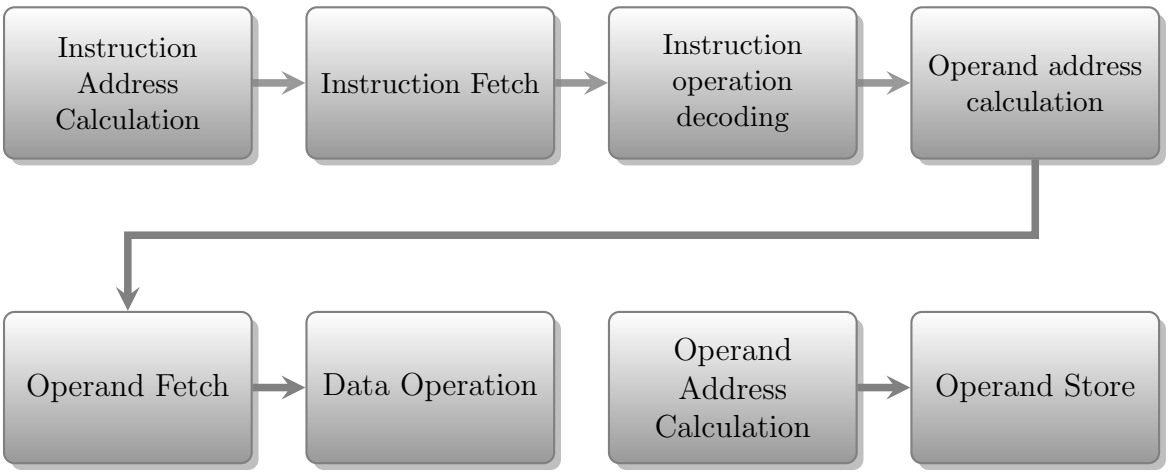
1. Processore-memoria: trasferimento dati dalla CPU alla Memoria R/W
2. Processore-I/O: trasferimento dati da CPU a I/O R/W
3. Elaborazione dati: operazioni logiche e aritmetiche sui dati operazioni della ALU
4. Controllo: può alterare la sequenza delle istruzioni, per esempio il salto

Esempio 3.
Parola = 16bit
Istruzione = 16bit
Codici operativi = 4 bit a sinistra = 2^4 combinazioni = 16

- 0001 carica in AC (accumulatore) una cella di M
- 0010 scrive in M il contenuto di AC
- 0101 somma una cella di M ad AC

1.4.3 Ciclo di esecuzione

1. Instruction Address Calculation
2. Instruction Fetch
3. Instruction operation decoding
4. Operand address calculation
5. Operand Fetch
6. Data operation
7. Operand address calculation
8. Operand Store



Per l'esecuzione di una singola operazione, bisogna prima eseguire una serie di altri sotto-compiti, che devono essere ben eseguiti.
Per questo è essenziale **l'unità di controllo**, per accertarsi che ogni operazione venga eseguita correttamente e nel giusto ordine.
La CPU può eseguire **più operazioni** momentaneamente, mantenendo ogni sua parte attiva , breve introduzione al concetto di pipeline

1.4.4 Interruzioni

Il meccansimo tramite il quale dei moduli possono interrompere la normale di sequenza di esecuzione.
Il ciclo di esecuzione con interruzioni è differente: al **termine** di un ciclo si ha il controllo delle interruzioni,

- se ce ne sono, esse vengono risolte
- se non ce ne sono, il ciclo ricomincia

Tipi di interruzioni:

- Program, overflow, divisione per zero
- Timer, da un timer inerno alla CPU
- I/O, termine di un'operazione di I/O
- Guasto Hardware

Si interrompe per

- efficienza elaborazione

Ciclo interruzione:

- viene aggiunto al ciclo di esecuzione
- la cpu controlla (fetch) le interruzioni pendenti
- se non ce ne sono, prende la prossima istruzione
- se ce ne sono:

- sospende esecuzione
- salva contesto
- imposta il pc all'indirizzo di inizio del programma di gestione
- esegue il programma di gestione dell'hardware
- rimette il contesto al suo posto e continua il programma interrotto

Lunghezza di attesa:

- **breve attesa**, tempo di operazione di I/O minore del tempo tra due istruzione WRITE
- **lunga attesa**,

Interruzioni multiple

In caso di **interruzioni multiple**: esistono vari livelli di interruzione e differenti tipi di politica di gestione.
Esistono interruzioni di alto livello e basso livello, a seconda dell'importanza che hanno per il funzionamento del sistema.
Politiche di interruzione:

- Disabilitare le interruzioni:
  - La CPU **ignora** le altre interruzioni e gestisce la prima e "maschera" le altre
  - Le interruzioni rimangono **pendenti**
  - vengono gestite nell'**ordine** in cui arrivano
- Definire le priorità
  - Interruzioni di bassa priorità vengono interrotte quando si presentano quelle di alta priorità
  - Quando è stata gestita la priorità di alto livello viene maneggiata quella di basso livello
  - Si hanno delle interruzioni **annidate**

1.5 Interconnessioni

Tutti i componenti **devono** essere connessi
Esistono vari tipi di connessioni per vari tipi di componenti, il bus collega:

- CPU
- Memoria
- I/O

Composto da 50 a qualche centinaio di linee

1.5.1 Bus

Tutti i dispostivi sono collegati dal bus di sistema
Il bus:

1. collega **2 o più** dispositivi
2. mezzo trasmissione condiviso
3. un segnale trasmesso ad un bus è disponibile a tutti i dispositivi
4. arbitro bus: solo un dispostivo alla volta può trasmettere
5. varie linee di comunicazione (trasmettono uno 0 o un 1)
6. varie linee trasmettono in parallelo numeri binari. Un bus da 8 bit trasmette un dato di 8 bit

Solitamente l'ampiezza del bus corrisponde ad un multiplo della parola.
Tipi di bus:

1. Bus di sistema:
   - connette cpu, i/o, M
   - da 50 a qualche centinaio di linee
   - 3 gruppi di linee
     (a) bus dati
     (b) indirizzi
     (c) controllo
2. Bus dati:
   - trasporta dati o istruzioni
   - ampiezza –> efficienza del sistema
     – con poche linee –> maggiori accessi in memoria
3. Bus indirizzi:
   - indica sorgente o destinazione dati
   - l'ampiezza determina la massima quantità di M indirizzabile

     Esempio 4.
     Architettura a 64 bit = $2^{64} - 1$ indirizzi
     Il bus deve essere di almeno 64 bit.

   - deve essere di almeno 1 parola
4. Bus controllo:
   Per controllare accesso e uso di:
   - **linee dati**
   - **indirizzi**
     (a) M write

(b) M read
(c) richiesta bus
(d) bus grant
(e) interrupt request
(f) clock

1.5.2 Uso del bus

Con modulo intendiamo una generica componente del computer.
se un modulo vuole inviare dati ad un altro:

- bus grant
- data transfer

se un module vuole ricevere dati da un altro:

- bus grant
- trasferire una richiesta all'altro modulo sulle linee di controllo
- attendere invio dati

1.5.3 Bus singoli e multipli

- singolo bus = ritardo e congestione
- vari bus = risoluzione problema

Esempio di Bus multiplo:

- **connessione punto a punto** fra CPU e cache, il che rende il trasferimento dei dati molto più veloce ed efficiente.

- **Bus di espansione**, si interfaccia con i dispositivi I/O
- **Bus ad alta velocità**, si interfaccia con i dispositivi I/O e fornisce una trasmissione di dati più rapida
- **Bus di sistema**, fra cache e Memoria principale.

1.5.4 Temporizzazione

Coordinazione degli eventi su un bus

- Asincrona
  - più complessa da implementare
- Sincrona
  - clock determined events
  - single clock line, with an alternate sequence of 0 and 1, with equal length
  - single 1-0 sequence is a clock cicle
  - every device connected to the bus can read the clock line
  - every event starts at the beginning of a clock cycle

QPI

Interconnessione Punto a Punto, con l'aumentare della velocità dei processori è sempre più frequenta una distribuzione di dati densa e veloce.
Un bus condiviso però rallenta la comunicazione del processore (bottleneck) , per questo si adottano delle connessioni punto a punto.

- Connessioni dirette multiple:
  più componenti del sistema godono di connessioni dirette a coppie con altri componenti.
- Architettura di protocollo a strati
- Trassferimento Dati a pacchetto:
  i dati non sono inviati come flussi di dati non elaborato ma come una sequenza di pacchetti, essi includono intestazioni di controllo e codici di correzione dell'errore.

Livelli di QPI:

- **fisico**
  la parte hardware, l'unità di trasferimento è di 20 bit (Phit) , tutto è sincronizzato da un clock.
- **Link**
  trasmissione affidabile del controllo del flusso, l'unità di trasferimento è di 80 bit (Flit).
  Protocollo con pacchetti da 72 (dati) + 8 (codice di correzione errore) bit.
  - Controllo del flusso:
    il mittente non può inviare più dati di quelli che il destinatario può ricevere.
  - Controllo dell'erroreç 8 bit per rilevare errori di trasmissione sugli altri 72 bit.
    In caso di errore il mittente deve re-inviare il pacchetto errato.
- **Routing**
  Struttura per dirigere i pacchetti attraverso essa. Determina il percorso che un pacchetto deve seguire.
  Supportato da tabelle di instradamento:
  - definite da software di basso livello che contiene delle istruzioni
  - descrizione percorso che un pacchetto può seguire
  - utile in sistemi di maggiori dimensione
- **Protocollo**
  Insieme di regole ad alto livello per lo scambio di pacchetti. Un pacchetto è un numero intero di Flits. Il contenuto di un pacchetto è flessibile, per gestire esigenze diverse.
  Supporta il protocollo di coerenza della cache, per garantire coerenza fra i contenuti della cache dei core e la memoria principale

1.6 Memorie

Caratteristiche principali della memoria:

- Locazione: processore (cache), interna (principale RAM), esterna (secondaria)
- Capacità: dimensione parola (dipende dall 'architettura), numero di parole
- Unità di trasferimento: parola, fra cache e RAM in blocco (insieme contiguo di parole)
- Metodo di accesso:
  - sequenziale: accedere prima a tutte le informazioni che vengono prima: lento. Utilizzato nei nastri magnetici, backup sistemi.
  - diretto, organizzata in gruppi, accesso sequenziale ai gruppi. accesso diretto ad un insieme di informazioni. HDD
  - casuale, accesso diretto a quella locazione di memoria, non importa dove si trova la locazione di memoria dell'informazione. RAM
  - associativo, informazione non indviduata da indirizzo, ma da una parte dell'informazione. CACHE
- Prestazioni: tempo di accesso, ciclo e velocità di trasferimento
- Modello fisico:
  - semiconduttore (ROM)
  - magnetico, campi magnetici (HDD)
  - ottico, laser ottico (CD)
  - magnetico-ottico
- Caratteristiche fisiche:
  - volatile (cache, quando termina il flusso di corrente elettrica i dati vengono cancellati)
  - non volatile (hdd, i dati vegnono memorizzati permanentemente quando si spegne il computer)
  - riscrivibile (RAM, HDD, cache)
  - non riscrivibile (ROM, read only memory).
- Organizzazione, se suddivisa in un singolo chip oppure vari, più o meno moduli.

Generalmente all'aumentare del costo della memoria aumenta la sua velocità ma diminuisce la sua ampiezza.
In ordine decrescente per velocità, costo e crescente per ampiezza:

- Registro (1 parola) 16 - 64 bit
- Cache (1 indirizzo)
- SRAM
- DRAM (8gb - 64gb)
- SSD (1TB - 8TB)

- HDD (16TB-24TB)
- CD - DVD-ROM
- Nastro (vari PB)

L'aumento di prestazioni delle CPU si deve a migliorie tecnologiche e architetturali, mentre quello delle memorie **solo** ad avanzamenti tecnologici.
Proprieta' dei programmi:

- Statiche, dal file sorgente
- Dinamiche, dall'esecuzione
  - Linearita' dei riferimenti, spesso consecutivi
  - Localita' dei riferimenti, indirizzi contigui sono piu' probabili

Definizione 13 (Congettura 90/10)**:**
Un programma impiega di solito il 90% del suo tempo di esecuzione alle prese con un numero di istruzioni pari a circa il 10% di tutte quelle che le compongono.

1.6.1 Gerarchia di memoria

Tutte le locazioni di memoria sono suddivise in blocchi.

Conviene organizzare la memoria in vari livelli gerarchici:

- Cache la più veloce e suddivisa in diversi livelli
  - L1 cache:
    molto veloce e molto costosa, per i dati ad accesso probabile
    es. 10% di cui parlavamo prima
  - L2 cache, più capiente ma meno veloce della L1.
  - L3 cache, più capiente ma meno veloce della L2
- Ram, più lenta della cache ma più capiente ed economica

La memoria Ram è composta da:

1. indirizzo di memoria
2. blocco di memoria

Memoria a livelli:

1. Livello inferiore, supporti con capacità più alti, piu' lenti, meno costosi
2. A livelli piu' alti diventano progressivamente piu' veloci, piu' costosi e meno capienti.

La CPU usa il livello piu' alto. Ogni livello inferiore contiene tutti i dati presenti ai livelli superiori.

trasferimento dati

I dati vengono scambiati sotto forma di WORD/PAROLE:

- CPU - Cache: scambio di PAROLE
- CACHE - MEMORIA P. scambio di BLOCCHI (multiplo di PAROLE)

La memoria lavora efficientemente se la CPU trova il dato cercato nella CACHE. Avendo piu' livelli di cache si ha una maggiore probabilità di trovare questo dato.

Il numero di parole in un blocco è una potenza di 2.
Una parola è composta da 4 byte, possiamo identificare i primi 14 bit come "indirizo" del bit, mentre i restanti 2 come identificativi del bit.

1.6.2 Cache

Un indirizzo di linea di cache e' costituito generalmente da:

1. Etichetta (tag), indirizzo del blocco nella memoria principale.
2. Blocco di K parole

Un blocco di memoria richiesto dalla CPU può essere presente **hit** o non presente **miss** in memoria. (generalmente è presente).
Per guadagnare efficienza prestazionale un **hit** deve essere molto probabile ($> 90\%$), se fosse minore di questa percentuale non avrebbe senso utilizzare quella struttura di memoria.
Un **miss** avvia una procedura di scambio di dati con un livello inferiore.
Una linea di cache può memorizzare diversi blocchi diversi, si usano i bit piu' a destra per identificare la parola all'interno della linea e i bit piu' a sinistra qual'e' la linea di cache per identificare il blocco.

Definizione 14 (Organizzazione)**:**
La memoria principale e' suddivisa in blocchi logici
La cache e' suddivisa in un multiplo di blocchi.

Definizione 15 (Tempo medio di Accesso)**:**
T_a*: Tempo medio di accesso ad un dato in memoria cache*

$$T_a \ = \ T_h \times P_h + T_m(1 - P_h) \tag{1.9}$$

T_h*: tempo di accesso ad un dato presnte in cache* T_m*: tempo medio di accesso ad un dato* ***non*** *in cache (dimensione blocco)* P_h*: probabilità di hit*

Tecnica generale

1. Suddivisione della memoria centrale in blocchi logici
2. dimensionamento della cache in multiplo di blocchi
3. ogni indirizzo emesso dalla cpu

- hit $\iff$ il dato viene fornito immediatamente alla cpu, (sotto forma di parola)
- miss, il dato viene fornito sotto forma di blocco
  (a) la cache richiede il dato al livello inferiore (memoria principale RAM)
  (b) viene posto in cache
  (c) viene fornito alla cpu

Definizione 16 (**associazione diretta / direct mapping**)**:**
Ogni blocco del livello ineriore può essere allocato solo in una specifica posizione ***linea/slot*** *del livello superiore*

1. ***ILS*** *= indirizzo di livello superiore*
2. ***ILI*** *= indirizzo di livello inferiore*
3. *$ILS = ILI \bmod N$, divisione con resto intero*

1. *vantaggi*
   - *semplicità traduzione indirizzo ILI a ILS*
   - *determinazione velocità hit o miss*
2. *svantaggi*
   - *necessità di contraddistinguere blocco in ILS*
   - *swap frequenti per accesso a dati di blocchi adiacenti,*
     il primo blocco di ogni insieme avrà la stessa linea di cache, quindi solo uno di questi puo' essere in cache

Definizione 17 (**associazione completa / fully associative**)**:**
Ogni blocco del livello inferiore può essere posto in qualunque posizione del livello superiore. Ad una cache di N blocchi viene associata una tabella di N posizioni contenenti il numero di blocco effettivo (tag)

- *vantaggi: massima efficienza di allocazione*
- *molto tempo per la corrispondenza ILS-ILI e della verifica hit/miss*
- *molto costoso dal punto di vista hardware e scarsa possibilità di hit*
- *difficile identificazione di hit o miss*

Definizione 18 (**associazinone a N-gruppi / N-way set associative**)**:**
Ogni blocco di un certo insieme di blocchi del livello inferiore può essere allocato liberamente in uno specifico gruppo di blocchi del livello superiore

Esempio 5.
Per una cache di 32 linee con un N equivalente a 2, ogni gruppo avrà 16 linee.
Ci sono 16 gruppi da 2 linee e per ogni coppia avremo un blocco di una determinata posizione di qualunque insieme.

Questo tipo di associazione è una via di mezzo fra gli altri due tipi. La cache composta da R gruppi di N posizioni di blocco, si affiancano R tabelle di N elementi contenenti i tag.
Per ottenere facilmente il numero di linee basta semplicemente fare la moltiplicazione

$$Numero\ di\ linee\ della\ cache = N \times R \tag{1.10}$$

Ha una buona efficienza di allocazione, nonstante abbia una certa complessità, e prende i due punti di forza degli altri due tipi:

- *uso efficiente delle linee di cache*
- *facile determinazione di hit o miss*

Non e' necessario avere un grande numero di N per raggiungere il massimo dell'efficienza1

1.6.3 Politiche di rimpiazzo dei blocchi

Quando si ha un miss, come si decide quale blocco della cache dobbiamo rimpiazzare?
Nell'associazione diretta non ci si pone questo problema, perchè ogni linea della cache corrisponde un blocco della memoria centrale.

1. *casuale*, viene occupato lo spazio omogeneamente, facile implementazione
2. *First-In-First-Out(FIFO)*, il blocco rimasto più a lungo in cache, complicata implementazione
3. *Least Frequently Used(LFU)*, il blocco con meno accessi, complicata implementazione hardware
4. *Least Recently Used(LRU)*, il blocco con l'accesso più distante, per preservare quelli accessi più recentemente, implementazione difficile.

A minor quantità di cache si hanno migliori prestazioni con il rimpiazzo LRU.
Questo e' il metodo piu' gettonato, e ad aumentare il livello di cache è sempre meno significativo il miglioramento offerto da queste tecnologie.
La scrittura dati determina incoerenza tra il blocco in cache e quello nei livelli inferiori

Definizione 19:
write through:

1. *scrittura contemporanea in cache e livello inferiore*
2. *aumento traffico per frequenti scritture nel medesimo blocco, dati coerenti fra blocchi*
3. *si ricorre a buffer asincroni verso la memoria, a causa di momenti di congestione del bus.*

La memoria contiene **istruzioni** e **dati** e solo il 50% delle operazioni sui dati sono scritture.

Definizione 20:
write back:

1. *scrittura in memoria inferiore differita al rimpiazzo del blocco di cache corrisp.*
2. *ridotto numero di modifiche nella memoria principale*

3. *occore ricordare operazioni di scrittura nel blocco*
4. *ottimizzazione del traffico tra livelli, riduzione congestione bus*
5. *periodi di incoerenza*

Occore ricordare che tra memoria centrale (RAM) e cache si passano **BLOCCHI** e non **PAROLE**.

Esempio 6 (scenario problematico).
- più dispositivi connessi allo stesso bus con cache locale
- memoria centrale condivisa

Nessun tipo di "write" (through, back) può assicurare coerenza.

Possibili soluzioni

- **monitoraggio del bus con write through**, controllori intercettano modifiche locazioni condivise
- **trasparenza hardware**, hardware aggiuntivo: modifica a RAM = modifica a cache
- **memoria non cacheable**, solo una porzione è condivisa e non cacheable

Cosa comporta la modifica di dati in una cache?

- invalida quella parola corrispondente nella memoria centrale
- invalida la parola nelle altre cache che la contengono
  per esempio in un sistema con CPU multi-core

Estendiamo il discorso alla memoria swap e RAM:
Quello che cambia e' che al livello cache-RAM e' tutto basato su un livello logico, mentre nell'altro caso si parla di un livello fisico fra RAM e memoria swap dei dispositivi di archiviazione eseterna.
Le problematiche che si incontreranno saranno le stesse: capienza RAM piena, politiche di rimpiazzo, modalita' di scrittura.
I blocchi diventano pagine di un programma.

Cache logica e fisica

Possiamo avere due tipi di Cache:

- Logica, riceve un indirizzo logico ma ad ogni cambiamento di contesto deve essere svuotata, conveniente se si prevede un utilizzo in un contesto con poche interruzioni
- Fisica, riceve un indirizzo fisico deve attendere la traduzione dell'indirizzo da logico a fisico, non deve essere svuotata ad ogni cambiamento di contesto, conveniente in un contesto con molte interruzioni

La cache deve essere svuotata ad ogni cambiamento di contesto, altrimenti non puo' funzionare bene.

Il problema dei miss

Esistono vari tipi di miss:

- di primo accesso, inevitabile non riducibile
- per capacità insufficiente, quando la cache non può contenere altri blocchi
- per conflitto, dipende dal tipo di associazione, quando vari blocchi possono corrispondere allo stesso gruppo

Soluzioni classiche:

- Maggior dimensione di blocco, aumento di miss, aumento linee utilizzo piu' efficiente dello spazio
- Maggiore associativita', incremento del tempo di localizzazione di un gruppo, soggetto alla regoal 2:1

Altre soluzioni:

- multilivello cache, fino a 3 livelli
- separazione cache dati / cache istruzioni
- ottimizzazione degli accessi mediante compilatori (C)

tipi di memorie a semiconduttore

Tipo	Categoria	Cancellamento	Mecc. scrittura	Volatile
RAM	read-write	elettric. byte-level	elettric.	si
ROM	read-only	non possibile	maschere	no
PROM	read-only	non possibile	elettric.	no
EPROM	read-mostly	luce UV, chip-level	elettric.	no
EEPROM	read-mostly	elettric. byte-level	elettric.	no
Memoria Flash	read-mostly	elettric. block-level	elettric.	no

Si dice di una memoria che e' volatile o no quando: se manca l'alimentazione l'intero contenuto della memoria si cancella.

Definizione 21 (DRAM)**:**
Dynamic RAM =

- *bit memorizzati in condensatori*
- *decadimento cariche con tempo*
- *refresh cariche, durante alimentazione*
- *semplice costruzione*

- *1 condensatore = 1 bit*
- *meno costose*
- *circuito per refresh*
- *meno veloci della SRAM*
- *usate nella RAM*
- *operazione analogica, la carica determina il valore (0 o 1)*

Definizione 22 (SRAM)**:**
Static RAM =

- *bit memorizzati tramite porte logiche*
- *non perde la carica*
- *no refresh*
- *piu' complesso, piu' elementi per bit (6 transistor)*
- *piu' costosa*
- *no refresh*
- *piu' veloce, utilizzata nella cache*
- *digitale*

1.6.4 ROM

Una ROM:

- memorizzazione permanente (non volatile)
- memorizzano:
  - microprogrammi
  - subroutine di libreria
  - programmi di sistema
  - funzioni tabulate (logaritmi, esponenziali, etc)

1.6.5 Codice correzione errore

Tipi di errori:

- Guasto hardware, non risolvibile
- errore software, possono accadere casualmente, a causa del decadimento della materia, i danni non sono permanenti

Quando un errore viene rilevato, puo' essere corretto attraverso i codici di Hamming (quelli che vedremo)
Quando un dato viene creato assieme ad esso si crea una sua 'impronta digitale' che verra' utilizzata per comparare la validita' del dato.
Si trasmettono gli 'M' bit di dati assieme a dei 'k' bit generati da una certa funzione che creano una sorta d'impronta digitale.
Attraverso la funzione che converte gli 'M' bit nei 'k' bit si utilizza per ricostruire il dato originale e confrontarlo con l'originale o lo si utilizza per correggerlo.
Questo tipo di codice di correzione è valido quando c'è un solo errore.
La formula per determinare la quantità di bit di correzione necessari:

$$2^{k-1} \geq M + K \tag{1.11}$$

Questa quantità diminuisce drasticamente all'aumentare degli 'M' bit:

Bit dati (M)	Bit controllo (k)	% incremento
8	4	50
16	5	31,25
32	6	18,75
64	7	10,94

Quando ci sono 2 errori si riesce a corregerne uno solo e si comunica che è stato rilevato un errore ma non è correggibile.

1.6.6 Memorie esterne

Le memorie esterne si suddividono in 4 macro-categorie:

- dischi magnetici
  - RAID
  - rimovibili
- dischi SSD
- ottica
  - CD-ROM
  - CD-Recordable (CD-R)
  - CD-R/W
  - DVD
- nastri magnetici

Dischi magnetici

I dischi sono ricoperti di materiali magnetici, e le informazioni è memorizzata in un campo **magnetico**.
Venivano costruiti in alluminio, mentre ora si utilizza il vetro perchè

- la sua superficie è più uniforme, (lo rende più affidabile),
- ci sono meno difetti di superficie (e ne riduce gli errori),
- è più rigido
- la testina può essere posta più vicino al disco
- è più resistente.

Il disco è organizzato in **cerchi concentrici** di informazione (chiamati tracce), il disco ruota e la **testina** viene spostata in senso radiale verso l'interno o l'esterno del disco.
Può leggere o scrivere rilevando **campi magnetici** o inducendoli. L'orientamento di un campo magnetico si determina in base al verso con cui la corrente scorre e determina il tipo di informazione che viene passata. Dato che i campi magnetici rimangono impressi nel materiale ferroso (ossido di ferro) è un dispositivo **non-volatile**.
L'interferenza fra campi magnetici è divisa fra le tracce da dei gap.
I dati vengono memorizzati tramite una bobina conduttiva di scrittura detta **testina**.
I dati sono memorizzati in anelli o **tracce concentriche**. I dischi ruotano ad una **velocità angolare costante**, ciò vuol dire che le tracce più interne a parità di rotazione perccoronon meno strada rispetto alle tracce più esterne.
Una conseguenza di questo fenomeno e del fatto che le informazioni contenute in ogni traccia devono essere della stessa quantità di byte, implica che i campi magnetici delle tracce più interne sarannno necessariamente più compatti. La densità delle tracce non è consistente.
Le tracce si dividono in:

- Settori, la dimensione minima di un blocco coincide con un settore
- Più di un settore per blocco
- Con più dischi, le tracce nella stessa posizione costituiscono un cilindro

Per **ricercare un settore** bisogna riconoscere l'inizio della traccia e del settore. Testina sulla traccia e attendere che il settore d'interesse passi attraverso la testina. (es. treno: quando prendiamo un treno ci sono 3 scenari: il treno sta per partire, dobbiamo aspettare un po', oppure il treno è già partito e dobbiamo aspettare il prossimo. Si può fare un'analogia con la testina (noi) settore (treno)). Il **formato di un disco** è:

- Informazioni aggiuntive non disponibili all'end user
- demarca tracce e settori

Esempio 7 (Disco Winchester).
In esso si possono ottimizzare la distanza fra traccia e testina.
In particolare un settore di questo formato è composto da 600B:

- GAP 1 = 17B
- Campo ID = 7B

  - Byte di sincronizzazione = 1B
  - N° traccia = 2B
  - N° testina = 1B
  - N° settore = 1B
  - Codice correzione errore = 2B
- GAP 2 = 41B
- Campo dati = 515B
  - Byte sincronizzazione = 1B
  - Dati = 512B
  - Codice correzione errore = 2B
- GAP 3 = 20B

Caratteristiche di un dispositivo di memorizzazione esterna:

- Testina
  - Fissa (raro)
  - mobile
- Disco
  - Rimovibile
  - Fisso
- Fascia
  - singola
  - doppia
- Piatto
  - singolo
  - doppio
- Meccanismo della testina:
  - contatto (floppy)
  - distanza fissa
  - seprazione aerodinamica
    * Testine foil planano sulla superificie dei dischi sfruttando la portanza del profilo
    * Testine vicinissime alla superficie dei dischi

Prestazioni

Le prestazioni di un disco eseterno sono determinate da:

- Tempo di posizionamento (T_P) (seek time), spostamento della testina nella traccia giusta, all'incirca dai 5 ai 20ms, non molto ottimizzabile essendo uno spostamento meccanico
- Latenza rotazionale (T_L) (latency), attendere che il settore d'interesse sia sotto la testina, dipende dalla velocità di rotazione (RPM)

$$RPM = 3600 \implies RPS = 60 \implies rotazione = 16.7ms \implies T_L = 8.35ms \quad (1.12)$$

- Tempo di accesso, ($T_P + T_L$) determinato dal tempo di posizionamento e dalla latenza (seek + latency)
- Tempo di trasferimento:

$$T = \frac{b}{r \times N} \quad (1.13)$$

dove **b** = byte da trasferire, **N** = numero di byte per traccia e **r** = velocità rotazione in secondi.

RAID

Redundant Array of Independent(/Inexpensive) Disks, esistono 7 livelli (da 0 a 6) non gerarchici di dischi, si tratta di un insieme di dischi fisici visti dal sistema operativo come un singolo dispositivo logico. In questo tipo di configurazione i dati sono distribuiti sui dispositivi.

RAID 0: nessuna ridondanza, tutti i dati sono distribuiti nei dischi in *strisce* (strip), si utilizza il metodo 'round robin striping' e si ottiene una maggiore velocità sia in scrittura che in lettura.
Le strip vengono memorizzate in parallelo nei vari dischi. Una strip è un multiplo di blocchi.
C'è un'alta probabilità che i dati siano distribuiti in vari dischi e quindi che non ci sia un conflitto di risorse.

RAID 1: Mirrored, il contenuto viene replicato su più dischi, ogni dato viene copiato in un altro disco. Ogni dato viene letto e scritto in più dischi. Il vantaggio è il recupero dei dischi immediato, è necessario solamente sosituire il disco rotto con quello che ha gli stessi dati. Si ha una copia esatta di ogni disco.

RAID 2: (non commercializzato), dischi sincronizzati, in modo tale che la testina di ogni disco sia nella stessa posizione in ogni disco, si usano unità di informazione molte piccole.
Si hanno codici di correzione calcolati tra bit corrispondenti nei vari dischi. Si ha una grande ridondanza, ciò fa aumentare di molto il costo.

RAID 3: simile al raid 2, ma ha solo un disco ridondante indipendentemente dal numero di dischi nell'array. Non è necessario sapere quale dato viene a mancare ma semplicemente quale disco non è più funzionante, quindi con questo sistema si sostituisce il disco guasto direttamente. Usa un semplice bit di parità per ogni insieme corrispondente di bit, attraverso questi si ricostruisce l'informazione e ai dati presenti negli altri dischi. Si ha un'alta velocità di trasferimento. La sincronizzazione dei dischi è causa di complessità, il disco di parità diventa il 'collo di bottiglia' per le operazioni in parallelo ed è quello più facile a subire un guasto

RAID 4: Ogni disco opera indipendentemente, ottimo per alti ritmi di richieste I/O, l'informazione di parità viene memorizzata su un disco ad hoc (disco di parità). Il problema di questa configurazione è il disco di parità che diventa un 'collo di bottiglia', essendo frequentemente utilizzato è quello più probabile a subire dei problemi di malfunzionamento

RAID 5: la parità viene distribuita in dischi diversi, non si ha più il collo di bottiglia dei RAID 3&4. L'allocazione viene distribuita con il metodo 'round robin', esattamente come i dati nel RAID 0, viene utilizzato nei server di rete, uguale al RAID 4 fuorchè per il disco di parità, robustezza fino ad un disco. Per avere 'N' dischi occorre averne 'N+1'

RAID 6: Si calcola la parità tramite due metodi distinti che sono memorizzati in blocchi separati in dischi differenti. Per avere 'N' dischi occorre averne 'N+2', si ha una robustezza fino a 2 dischi, equivale al RAID 5 con il doppio della parità.

Dischi SSD

Solid State Drive, Solid perchè si basa su circuiti integrati ovvero **memorie flash** di tipo NAND
Il **floating gate**

- non attivo, non interferisce con il control gate e rappresenta bit a 1, di default è in questo stato.
- se attivo, tramite alto voltaggio intrappola elettroni che rimangono anche in assenza di alimentazione e rappresnetano bit a 0 e rende il transistor come se fosse inutilizzabile

La struttura di una memoria flash di tipo NAND è organizzata in arraya da 16 o 32 transistor collegati in serie. La bit line va a 0 solo se tutti i transistor delle corrispondenti linee della parola sono a 1 (attivi), deriva dalla funzione booleana NAND. Le letture e le scritture coinvologono l'intera parola.
I vantaggi di questi dischi sono:

- velocità, non sono coinvolte operazioni meccaniche.
- alte prestazioni di I/O e aumenta le prestazioni dei sottoinsiemi di I/O
- durata, meno suscettibile a urti e vibrazioni rispetto ai dischi magnetici
- maggiore durata, non soggetti a usura meccanica
- consumo energetico inferiore, meno energia non essendo coinvolte parti meccaniche.
- funzionalità più silenziose e fredde, minori costi energetici
- tempo di accesso e latenza inferiori oltre 10 volte rispetto a quelli degli HDD

Organizzazione di un disco SSD:

- Sistema host:
  1. per accedere ai dati, il sistema operativo richiama il software del file system, che richiama a sua volta il software del driver di I/O che fornisce l'accesso all'SSD
  2. il componente di interfaccia si riferisce all'interfaccia fisica ed elettricatra il processore host e l'SSD
- SSD:

1. Controller: fornisce l'interfacciamento a livello del dispositivo SSD e l'esecuzione del firmware
2. Indirizzamento: logica che esegue la funzione di selezione tra i componenti della memoria flash
3. Buffer/cache dati: RAM ad alta velocità per compensare velocità e aumentare il throughput dei dati
4. correzione degli errori: logica per il rilevamento e la correzione degli errori
5. Componenti della memoria flash: singoli chip flash NAND

Gli svantaggi degli SSD:

- Le performance decadono con l'uso, la dimensione di un blocco delle memorie flash all'interno degli SSD è di solito di 512kb, per poter scrivere anche solo un B bisogna portare il blocco in cache e modificare l'intero blocco riscrivendolo da capo.
  I file sono di solito salvati in pagine da 4kB quindi 128 pagine per blocco. con l'utilizzo dei file essi si frammnetano, le pagine vengono memorizztei in blocchi diversi e le prestazioni decadono.
  soluzioni: over-provisioning, vengonon tenuti certi blocchi solamente per le scritture; cancellazione delle pagine inattive; comando TRIM (avverte il dispositivo quali pagine sono state memorizzate logicamente).
- si ha un numero limitato di scritture, intorno alle 100.000, per risolvere questa limitazione, cache front-ending (ospita blocchi più riferiti, politica LRU), distribuzione scritture, gestione blocchi esauriti, RAID.

Memorizzazione Ottica

Inizialmente furono concepiti per organizzare dati audio: 650MB memorizzano più di 70 minuti audio.
Sono principalmente dischi di policarbonato rivestiti con un materiale altamente riflettente.
I dati sono memorizzati come microscopici pozzetti:

- Land
- Pit

i dati 0 e 1 vengono memorizzati come transizione da land e pit o viceversa e vengono letti tramite laser.
Si ha una densità di memorizzazione costante, infatti si ha una singola traccia organizzata a spirale e gira a velocità costante, la velocità del disco dipende dalla posizione in cui si trova.

CD-ROM

Caratteristiche:

- Audio: singola velocità
  - velocità lineare costante
  - 1.2 ms^{-1}
  - traccia a spirale lunga 5.27km

  - memorizza 4391 secondi = 73.2 minuti
- la velocità dichiarata è la massima raggiungibile che il lettore può raggiungere.
- formato dati:
  - modo 0 = campo dati vuoto
  - modo 1 = 2048 byte dati+correzione
  - modo 2 = 2336 byte dati

Accesso casuale su CD-ROM:

- difficile causa della velocità lineare costante
- spostare la testina in posizione approssimata
- configurare la giusta velocità di rotazione
- legere l'indirizzo
- altri aggiustamenti per spostarsi sul settore richiesto

Non è però efficiente come in una memoria RAM
Pro:

- Capacità, si utilizzavano come sistemi di bakcup, erano più economici degli HDD
- facili da produrre su grande scala
- rimovibile
- robusto

Contro:

- Costoso per piccole quantità
- lento per accedere ai dati
- essendo un CD-ROM è di solo lettura

1.7 Formulario

Altri tipi di memorizzazione ottica

- CD-R(ecordable)
- CD-RW
- Digital video Disk usato per riprodurre film,
  sono presenti molteplici strati,
  ha un'alta capacità di memorizzazione
  si usa tutto lo spessore del disco e inoltre si usano entrambi i lati di esso,
  il diametro del raggio laser è minore quindi si ha una maggiore densità di memoria

1.7.1 Nastro magnetico

- Accesso seriale
- lento
- molto economico
- utilizzato per backup e copia di riserva

Le informazioni possono essere memorizzate a serpentina, o possono essere scritte a blocchi

FINE PRIMA PARTE.

Chapter 2

Seconda Parte

2.0.1 Input/Output

Un computer è collegato a innumerevoli apparecchi i quali

- gestiscono quantità di dati differenti
- velocità diverse
- formati diverse

È certo che siano più lenti della CPU e della RAM, per ottenere il massimo delle prestazioni della propria CPU è necessario che le periferiche esterne siano controllate da dei moduli comuni: **moduli di input output**.

Vari tipi di dispositivi esterni:

- Comprensibili dall'uomo
  Monitor, stampanti, ...
- Comprensibili dalla macchina
- Comunicazione
  Modem, router

I componenti di un dipositivo esterno:

- Logica di controllo
  Comunica con il modulo di I/O
- Buffer di memoria
  immaggazzina memoria, esempio (logistica container)
- Transducer
  trasforma dati fisici (analogici) in bits o viceversa

Componenti di un modulo I/O:

-

2.0.2 Tecniche di gestione input/output

I/O da programma

Attesa attiva della CPU e si occupa del trasferimento dei dati I/O memory mapped o separato

I/O guidato da interrupt

Non c'è l'attesa della CPU, essa viene interrota quando il dispositivo è pronto

Direct Memory Access

Hardware aggiuntivo
Modulo DMA:

- Data Lines
  - Data Count
  - Data Register
  - Address Register
- Address Lines
  Address Register
- Control Logic

Operazioni DMA:

- CPU comunica al controllore DMA:
  - lettura scrittura
  -
-
-

Il trasferimento dei dati DMA occupa il bus e lo sottrae alla CPU, si possono utilizzare due metodi differenti per accedere al canale:

- Una parola alla volta, cycle stealing, sottrae di tanto in tanto per un solo ciclo
- Per blocchi, burst mode, è il metodo più efficace in quanto l'occupazione del bus è un'operazione molto onerosa.

a

2.1 Rappresentazione binari numeri reali

2.2 Linguaggio Macchina

Il linguaggio macchina è costituito da:

- Insieme istruzioni eseguibili dalla CPU
- Tipologia dei dati manipolabili
- Tipi di operandi

Nessun linguaggio macchina è superiore ad un altro, ed esso dipende dall'ambito applicativo e dalle funzioni che si devono far svolgere ad esso.
Gli elementi dell'istruzione della macchina:

- Codice operativo, specifica l'operazione da eseguire
- Riferimento all'operando sorgente, specifica l'operando che rappresenta l'input dell'operazione
- Riferimento all'operando risultato, dove va messo il risultato ottenuto
- Riferimento all'istruzione successiva

Gli operandi si possono trovare:

- Memoria centrale, o (virtuale)
- Registri della CPU, ognuno ha il proprio numero identificativo
- Dato immediato con l'istruzione
- Dipsositivi di I/O

L'istruzione è una sequenza di bit divisa in campi, viene usata una rappresentazione simbolica delle configurazioni di bit. Pure gli operandi hanno la propria rappresentazione simbolica
Il formato di un'istruzione di 16 bit è:

- 4 bits, Codice operativo (opcode)
- 6 bits, Indirizzo operando, (operand reference)
- 6 bits, Indirizzo operando, (operand reference)

I vari tipi di istruzioni sono di:

- elaborazione dati, istruzioni aritmetico logiche, nei registri della CPU
- Immagazinamento dei dati in Memoria o recupero dati dalla Memoria
- Trasferimento dati (I/O)
- Controllo del flusso del programma

Gli indirizzi necessari per un'istruzione possono essere:

- un indirizzo per ogni operando (1 o 2)
- uno per il risultato
- indirizzo dell'istruzione successiva

Quindi possono essere al massimo 4, (cosa molto rara e dispendiosa), ma in genere sono 1, 2 o 3 per gli operandi/risultati
Quando si riferisce ad un solo registro vuol dire che il secondo è implicito ed è memorizzato in un registro, per esempio nell'accumulatore.
Quando non si riferisce a nesssun indirizzo vuol dire che tutti gli indirizzi sono impliciti e si sta utilizzando una pila in cui si accumulano i vari indirizzi utilizzati.

- Se si utilizzano **meno indirizzi** di conseguenza si eseguiranno delle istruzioni più elementari e corte, se si hanno più istruzioni per uni stesso programma porterà ad un tempo di esecuzione più lungo. L'archiettura RISC utilizza questa filosofia, significa infatti **Reduced Instruction Set Computer** e si basa sul velocizzare le istruzioni più frequenti.
- se si utilizzano **più indirizzi** le istruzioni diventeranno più complesse quindi impiegherà meno tempo per eseguire istruzioni più complesse ma userà più potenza per istruzioni semplici.
  L'architettura RISC si basa su questo principio **Complex Instruction Set Computer**

Cosa comporta progettare un insieme di istruzioni:

- Repertorio:
  quante e quali operazioni
- Tipo di dato:
  su quali dati
- Formato:
  lunghezza, numero indirizzi, dimensione dei campi
- Registri:
  numero di registri della CPU indirizzabili dalle istruzioni
- Indirizzamento:
  modo di specificare gli indirizzi degli operandi

Tipi degli operandi

- Indirizzi, rappresentati come interi senza segno
- Numeri
  limite al modulo
  limite alla precisione
- Caratteri (stringhe)
- Dati logici, variabili booleane per il controllo del flusso dell'esecuzione.

I numeri vengono rappresentati:

- Interi (con la virgola fissa)
- Virgola Mobile (Floating point, IEEE754)
- Decimali impaccati, nelle operazioni di I/O, non efficienti in quanto per rappresentare una cifra decimale si utilizzano 4 bit, che vuol dire solo 10 delle 16 configurazioni disponibili vengono utilizzate, si utilizza per evitare la conversione.
  es: 246 viene rappresentato come:

$$246 = \underset{200}{0010}\ \underset{+40}{0100}\ \underset{+6}{0110} \tag{2.1}$$

I caratteri vengono rappresentati in ASCII (American Standard Code for Infromation Exchange) da 7 bit, quindi si hanno in totale 128 configurazioni disponibili.
Si ha di solito 1 bit per i caratteri di controllo.
C'è inoltre una versione estesa da 8 bit in cui si possono rappresentare 256 configurazioni.
Dati Logici:

- n bit, invece che un singolo dato
- manipolare bit separatamente

Tipi di dati Intel x86

- 8 (byte), 16 (word), 32 (doppia parola), 64 (quadword), 128 (double quadword) bits
- L'indirizzamento è per unità di 8 bit (1 byte)
- Una double word da 32 bit inizia da un indirizzo divisibile per 4
- Non si ha la necessità di allineare gli indirizzi per le strutture dati in memoria
- Bisogna allineare i dati per i trasferimenti dati nel bus

Ci sono vari tipi di operazioni:

- Trasferimento Dati,
  comporta specificare: la sorgente (dove si trova il dato), la destinazione (dove andrà messo il d.), la lunghezza del dato da trasferire
- Aritmetiche,
  somma, sottrazione, moltiplicazione, divisione.
  I numeri interi hanno sempre il segno, viene utilizzata anche per numeri con la virgola mobile e inoltre possono esserci le operazioni di:
  incremento (+1), decremento(-1), negazione (inversione del segno, trovare il numero opposto), calcolo valore assoluto
- Logiche,
  sono operazioni dirette sugli specifici bit, operazioni di: AND, OR, NOT, XOR, EQUAL, possono anche essere eseguite parallelamente su tutti i bit di un registro
- Conversione
- I/O
- Sistema
- Trasferimento del controllo,
  Salto condizionato (branch), per esempio BRE R1, R2, X (salta a X se R1 equivale a R2), si ha la necessità di saltare se si deve eseguire più volte una stessa operazione come in un 'for, while' loop, il che dà spazio alla programmazione modulare.
  Salto incondizionato, salta alla prossima istruzione, non ha operandi in quanto non si devono verificare delle condizioni.
  Chiamata di procedura: una proc. è un pezzo di programma a cui si può dare un nome il che permetto di eseguirlo indicandolo con il nome. Questo permette di risparmiare di scrivere codice e di poter affidare a qualcun'altro la scrittura di questo. Dobbiamo avere due istruzioni: la chiamata e il ritorno.
  Al fine di memorizzare l'indirizzo di ritorno ci sono 3 luoghi di memorizzazione differenti:

- Registro, non funzionale quando sono presenti dei cicli ricorsivi (ovvero che si 'richiamano')
- All'inzio delle procedura chiamata, non funzionali sempre in presenza di cicli ricorsivi
- Cima della pila, ovvero si utilizza una porzione di M dove le scritture e le letture avvengono sempre in cima. In questo modo si richiamano gli indirizzi che sono stati per ultimi scritti nella pila e che si trovano appunto in cima, evitando il problema dei loop ricorsivi.

2.2.1 Linguaggio Assembly

Questo linguaggio è ad un livello più alto rispetto al linguaggio macchina ed è più comprensbile dall'uomo.
Gli indirizzi numerici (binario) vengono interpretati come indirizzi simbolici (A, B, ...,Z),
I codici operativi diventano simboli (SUB, ADD, BRE).
L'assemblatore è un programma che traduce dal linguaggio assembly al linguaggio macchina.

Big / Little Endian

I bit nella memoria vengono memorizzati in maniera differente in base all'architettura del calcolatore, ovvero:

- Big Endian,
  I bit più significativi vengono memorizzati prima negli indirizzi, da sinistra a destra
- Little Endian,
  I bit meno significativi vengono memorizzati negli ultiimi indirizzi, da destra a sinistra.

2.2.2 Modi di indirizzamento

Esistono diversi tipi per specificare l'indirizzo degli operandi:

- Immediato
- Diretto
- Indiretto
- Registro
- Registro indiretto
- Spiazzamento
- Pila

Immediato

L'operando è specificato nell'istruzione stessa, (nella parte del campo indirizzo)
VANTAGGIO: non si esegue nessun accesso in Memoria (operazione molto onerosa, in quanto implica l'occupazione del bus)
SVANTAGGIO: limitato dalla dimensione del campo indirizzo.
Se abbiamo per esempio 6 bit destinati al campo indirizzo, avremo 2^6 valori diversi.

Diretto

Il campo indirizzo contiene l'indirizzo dell'operando in Memoria.
SVANTAGGIO: un accesso in Memoria (operazione onerosa) e, spazio di indirizzamento limitato legato alla grandezza della memoria.

Indiretto

Il campo indirizzo contiene l'indirizzo di una cella di Memoria che contiene l'indirizzo dell'operando.
VANTAGGIO: Con parole di lungehzza N si possono indirizzare 2^n entità diverse, n deve essere comunque uguale o inferiore alla grandezza del campo indirizzo.
SVANTAGGIO: è che si avrà bisogno di 2 accessi alla memoria.

Registro

L'operando si trova in un registro indicato nel campo indirizzo
VANTAGGIO: pochi bit per l'indirizzamento.
SVANTAGGIO: limitato dal numero di registri disponibili dal tipo di architettura.

Registro Indiretto

Si basa sullo stesso principio dell'indirizzamento a registro, l'operando si trovan in una cellla di Memoria puntata dal contenuto del registro.
VANTAGGIO: si ha solo 1 accesso in memoria a differenza dell'indirizzamento indiretto, si ha anche un grande spazio di indirizzamento (che dipende dal numero di registri e dalla lunghezza del campo indirizzo)

Spiazzamento

È la combinazione dell'indirizzamento diretto e a registro indiretto. Il campo indirizzo è suddiviso in due parti:
il valore di base (A)
il registro che contiene l'indirizzo di un valore da sommare ad A
Una versione di questo tipo è l'indirizzamento **relativo**, in cui non si ha un registro casuale ma il Program Counter.
L'indirizzamento **registro-base**, in esso A contiene lo spiazzamento, R contiene il puntatore all'indirizzo base.
L'**indicizzazione** ha in 'A' l'indirizzo base e nel campo registro lo 'spiazzamento', per indicare gli operandi da un certo punto della memoria in poi indicheremo con 'A' questo punto di partenza e per accedere a tutti i dati successivi basta incrementare il contenuto del campo registro di 1.

2.2.3 Stack/Pila

La pila è una sequenza lineare di locazioni riservate della Memoria, c'è un puntatore (che si trova nel registro SP, stack pointer, ha come indirizzo la cima della pila). L'operando si trova nella cima della pila, si può considerare come un'evoluzione dell'indirizzamento a registro indiretto.

2.2.4 Formato delle istruzioni

Il formato delle istruzioni è come sono scritte le istruzioni date alla macchina, influsice la struttura dei campi dell'istruzione, include il codice operativo, include uno o più operandi e

generalmente si ha più di un formato per linguaggio macchina.

Lunghezza delle istruzioni

È strettamente correlata al formato delle istruzioni, è influenzata e influenza:

- La dimensione della Memoria
- L'organizzazione della Memoria
- Struttura del bus
- La complessità della CPU
- La velocità della CPU

Per sfruttare al meglio tutte le risorse di un calcolatore deve essere un giusto compromesso fra un repertorio delle istruzioni potente e la necessità di risparmiare spazio (fisicamente nella parte hardware della macchina).

Esiston due tipi di formati delle istruzioni:

- Lungehzza fissa (fixed length),
  esempio: PDP-8, PDP-10
- Formati a lunghezza variabile o ibrida (hybrid/variable length),
  esempio: PDP-11, VAX, Intel x86, questo tipo aggiunge complessità al realizzazione della macchina, ma può renderla più potente per certi utilizzi specifici.

Allocazione dei bit

Come i bit vengono allocati dipende dai diversi tipi di indrizzamnento usati da una determinata architettura, dal numero variabile degli operandi (0, 1, 2), il numero dei registri, dei banchi registri (tipi di 'buffer' di memoria). Dipende anche dall'intervallo degli indirizzi (ogni quanto viene ripetuto un indirizzo) e dalla loro granularità (se sono a byte o parola), l'indrizzamento a byte è più oneroso ma utile per la manipolazione dei caratteri.

2.2.5 Approfondimento sul funzionamento della CPU

La CPU, essendo il componente hardware più potente della macchina è la parte più importante, essa ha i compiti di:

- Prelevare le istruzioni (Instruction Fetch)
- Interpretare le istruzioni (Instruction Decode)
- Prelevare Dati (Operand Fetch)
- Elaborare Dati (Execute)
- Scrivere Dati (Write Back)

Le componenti principali della CPU sono:

- ALU, l'unità aritmetico logica, il 'cervello' della CPU

- Registri, la memoria della CPU
- Unità di Controllo, le parti che verificano il corretto funzionamento del processore

La CPU è inoltre collegata al bus di sistema per poter interagire con gli altri componenti dell'elaboratore

Registri

I registri sono uno 'spazio' di lavoro in cui la CPU può memorizzare i dati che ha elaborato senza dover interagire con la Memoria Principale, in quanto questa è un'operazione molto onerosa e che si deve cercare di limitare al più possibile.
I registri sono al vertice della gerarchia di memoria. Essi possono avere funzioni diverse determinate dall'impianto progettuale della CPU.

I tipi di Registri sono:

- Utente, vengono utilizzati dal 'programmatore' per memorizzare internamente i dati alla CPU e successivamente da elaborare.
- di Controllo e di Stato, utilizzati dall'unità di controllo per monitorare le operazioni svolte dalla CPU, sono anche utilizzati dai programmi del sistema operativo per controllare l'esecuzione dei programmi.

Con 'programmatore' ci si riferisce a:

- L'umano che programma in assembly, (che poi viene trasformato in codice macchina dall'assemblatore)
- Il compilatore che produce un codice in assembly da un programma in HLL (Linguaggio ad Alto Livello)

I registri che sono visibili all'utente sono:

- Uso generale (general purpose),
  possono essere ad uso generale o dedicati a particolari funzioni, possono memorizzare indirizzi, dati
- memorizzazione dei dati
- memorizzazione di indirizzi
- memorizzazione dei codici di condizione

Inoltre la memoria principale può essere organizzata logicamente come un insieme di 'segmenti' (spazi di indirizzamento multipli).
Un segmento al suo interno contiene locazioni di memoria indirizzabili, inoltre si può indicare all'interno della memoria fisica la 'base' del segmento (dove comincia) e la sua 'lunghezza' (quanto è lungo effettivamente)

Registri ad uso generale: Questi registri si possono dividere in due sottocategorie:

- Effettivamente ad uso generale,
  aumentano la flessibilità e le opzioni disponibili al 'programmatore',
  aumentano le dimensioni dell'istruzione e della sua complessità, in quanto necessitano di uno spazio separato nel formato dell istruzioni che può variare la lunghezza in base al numero di registri.
- Specializzati,
  le loro istruzioni sono più piccole e veloci,
  a discapito di un'inferiore flessibilità

In genere il numero dei registri generali varia da 8 a 32, se ce ne fossero meno di 8 si avrebbe un maggior numero di accessi in memoria principale (operazione onerosa), se fossero più di 32 non si limiterebbe l'accesso alla memoria e aggiungerebbe molta complessità alla struttura della CPU.
Tuttavia l'architettura RISC arriva ad utilizzare fino a centinaia di registri.

Lunghezza dei registri: Generalmente un registro è lungo abbastanza da poter contenere un indirizzo della memoria principale e da contenere una 'full word'

Registri per memorizzazione di Codici di Condizione: Si tratta dell'insieme di bit individuali che possono essere letti implicitamente da un programma non possono essere impostati da un programma

Registri di Controllo e di Stato: Sono registri che abbiamo già incontrato:

- Progrma Counter (PC), il registro ceh tiene il 'conto' delle istruzioni da svolgere
- Instruction Register (IR), il registro che contiene le istruzioni da eseguire
- Memory address register (MAR), il registro degli indirizzi di memoria
- Memory Buffer Register (MBR), un registro di buffer, ovvero una memoria temporanea interna al processore

Program Status Word: La PSW è un insieme di bit che include codici di condizione fra cui:
il segno dell'ultimo risultato, zero, riporto, uguale, overflow, abilitazione/disabilitazione interrupt, supervisore

Modo Supervisore: Si tratta di una modalità che permette al Sistema Operativo di utilizzare le procedure del Kernel, che agiscono su componenti critiche del sistema, ovvero l'esecuzione di istruzioni privilegiate. Essa è disponibile SOLAMENTE al sistema operativo e non all'utente o al programmatore in assembler.

Ciclo di esecuzione CPU con indirettezza:

Al ciclo di esecuzione della CPU nella fase di fetch e di execute hanno un sottociclo che è l'indirettezza.
Al fine di recuperare gli operandi indicati in un'istruzione può essere necessario accedere più volte in memoria secondo la modalità di indirzzamento indiretto.

Flusso dei dati L'instruction Fetch generalmente si suddivide in queste operazioni:

- L'indirizzo dell'istruzione successiva 'x' è contenuta nel PC;
- L'indirizzo dell'istruzione 'x' viene spostato nel MAR;
- L'indirizzo 'x' viene emesso nel bus indirizzi;
- L'unità di controllo richiede una lettura nella memoria principale;
- Viene letto l'indirizzo nella memoria principale;
- L'indirizzo ottenuto viene inviato tramite il bus dati e viene ricevuto e copiato dal MBR;
- L'indirizzo viene spostato nell'IR
- Viene incrementato il PC con l'indirizzo dell'istruzione 'x+1'

Data Fetch Il ciclo del data fetch comprende:

- Viene esaminato l'IR;
- Se il codice operativo (opcode) dell'istruzione richiede un indirizzamento indiretto si esegue il ciclo di indirettezza:
  - N bit più a destra del MBR vengono trasferiti nel MAR;
  - L'unità di controllo richiede la lettura dalla memoria principale;
  - Il risultato della lettura, l'indirizzo dell'operando, viene trasferito nel MBR;

Execute Questa parte del ciclo può variare molto in base al tipo di architettura, dipende dal tipo di istruzioni che bisogna eseguire e può includere le operazioni di:

- Lettura/scrittura della Memoria;
- Input/Output;
- Trasferimento di dati fra registri e/o nei registri;
- Operazioni della ALU;

Interrupt Esso è semplice e prevedibile si svolge come segue:

- Viene salvato il contenuto del PC, al fine di recuperare il ripristino dell'esecuzione dopo la gestione dell'interruzione;
- Il PC viene caricato con l'indirizzo della prima istruzione della routine di gestione dell'interruzione;
- Il fetch dell'istruzione viene puntato al PC;

Prefetch È la fase di prelievo dell'istruzione e accede alla memoria principale, quest'istruzione viene di solito eseguita durante la fase di esecuzione dell'istruzione corrente (durante questa fase non si deve accedere in memoria).
Questa procedura può diventare inutile se sono in esecuzione delle istruzioni di 'jump' o 'branch' che vanno a modificare il contenuto delle istruzioni da eseguire.

2.3 Pipeline

Il concetto di **pipeline** è quello di suddividere il lavoro e:

- eseguire più attività contemporaneamente
- eseguire il lavoro in un tempo minore

Tuttavia non si può raggiungere il parallelismo totale come conseguenza della **dipendenza funzionale**, ovvero quando si necessità che l'attività precedente sia terminata per poter accedere ai dati che ha elaborato.
Nella pipeline i lavori sono affidati a degli **esecutori** che possono di diversi tipi:

- generici, ogni esecutore può eseguire un lavoro completo, ogni esecutore ha le stesse risore, ha lo stesso throughput del parallelismo totale
- specializzati, ogni esecutore svolge sempre la stessa fase, ogni esecutore è limitato alla propria fase, ogni lavoro passa da un'esecutore all'altro, questo tipo utilizza meno risorse.

Il secondo metodo è quello più diffuso in quanto:

- Ogni lavoro viene suddiviso in certo numero di fasi (i)
- Ogni fase è svolta da operatori diversi
- In ogni istante sono eseguite fasi diverse
- In ogni fase successiva ogni operatore riesegue la stessa fase.

Le fasi del ciclo esecutivo di un'istruzione sono:

- prelevare istruzione
- interpretare istruzione
- prelevare dati
- elaborare dati
- memorizzare dati

Ognuna di queste fasi è eseguita da una diversa unità funzionale della CPU, per esempio l'ALU esegui la fase di elaborazione dati.
Al fine di non perdere nessuna informazione e senza dover utilizzare la memoria principale si utilizzano dei buffer (registri temporanei), su cui si scrivono i dati utili alla fase successiva. Ne viene posto uno fra ogni fase.
Per aumentare le prestazioni bisogna:

- decomporre maggiormente il lavoro
- rendere le fasi più indipendenti fra loro e con una durata simile

Tuttavia aggiungere più fasi alla pipeline può essere una fonte di criticità perchè si creano più dipendenze e più possiblità di malfunzionamenti.

2.3.1 Pipeline hazards

Una situazione di criticità della pipeline in cui:
l'istruzione successiva non può essere eseguita nel ciclo di clock immediatamente successivo (stallo).

1. sbilanciamento delle fasi
2. problemi strutturali
3. dipendenza dai dati
4. dipendenza dal controllo

Sbilanciamento delle fasi Questa criticità si ha in quanto non tutte le fasi richiedono lo stesso tempo di esecuzione.
per esempio: la lettura di un operando tramite registro o mediante indirizzamento indiretto.
Per questo motivo la suddivisione in fasi va misurata in base alla durate dell'istruzione più onerosa.
Inoltre non tutte le istruzioni richiedono le stesse fasi e le stesse risorse.
Il rischio che si corre con questo tipo di problema è che se un esecutore ha svolto un lavoro e deve passare un dato al successivo esecutore finchè quest'altro sta ancora svolgendo la propria mansione rischia di sovrascrivere i dati di quest'ultimo.
Come soluzione a questo tipo di problema ci sono due possibli soluzioni:

- decomporre le fasi più onerose in più sottofasi,
  ha però un elevato costo e una scarsa utilizzazione.
- duplicare gli esecutori delle fasi più onerose e farli operare in parallelo,
  per esempio due ALU, una per l'aritmetica intera e una per l'aritmetica a virgola mobile.

Problemi strutturali Accadono quando due o più istruzioni che sono già nella pipeline richiedono di accedere ad una stessa risorsa nello stesso ciclo di clock. In questo caso gli accessi devono essere sequenziali e non paralleli.
Per risolvere questo tipo di problema si attuano queste strategie:

- Introduzione fasi non operative (nop)
- suddivisione delle memorie in modo tale che permettano gli accessi paralleli:
  per esempio una cache per le istruzioni e una per i dati.

Dipendenza dai dati Accade quando una fase non può essere eseguita in un certo ciclo di clock perchè i dati di cui ha bisogno non sono ancora disponibili, ovvero deve attendere il termine dell'elaboraione di una fase precedente.
Avendo due istruzioni: istruzione i, istruzione i+1, si possono avere questi tipi di criticità:

- **read after write**: lettura dopo scrittura,
  i+1 legge prima che i abbia scritto,

  1. Si introducono fasi non operative (nop) quindi fasi di stallo

2. si propagano i dati appena sono stati calcolati, (dataforwarding),
   se l'operando viene calcolato alla fine della fase *EI* lo si comunica subito doppo questa fase (e non nella fase successiva *WO*) e si riduce di uno le fasi di stallo.
3. Si riordinano le istruzioni in maniera più efficiente dal compilatore.

- **Write After Write** : scrittura dopo scrittura,
  i+1 scrive prima che i abbia scritto
- **Write After Read** : scrittura dopo lettura,
  i+1 scrive prima che i abbia letto (caso raro in pipeline)

Dipendenza dal controllo Accade quando entra nella pipeline un'istruzione di salto condizionato o incodizionato.
Nel caso del salto condizionato:

- Si mette in stallo la pipeline finchè non si conosce l'esito della condizione del salto
- Si individuano le istruzioni critiche e si aggiunge un'apposita logica di controllo, nel farlo si complica il compilatore e l'hardware specifico
- Si aggiungono flussi multipli, ovvero si caricano le due possibili istruzioni che sono specificate nell'istruzione di salto:
  l'istruzione *n* o l'istruzione *i+1* .
- Si esegue il prefetch dell'istruzione target del salto.
- Buffer circolare (*loop buffer*),
  ha una capienza di 256 bytes, viene indirizzato al byte, dato un indirizzo di target controllo se c'è nel buffer: 8 bit meno significativi (come indice buffer), gli altri bit più significativi si utilizzano per verificare se la destinazione del salto sta già nello sticker
  - Piccola e veloce memoria che ricorda tutte le ultime n instruzioni prelevate
  - In caso di salto si controlla se l'istruzione è già dentro il buffer
  - Utile in caso di loop, se il buffer contiene tutte le istruzioni da eseguire nel loop esse devono essere caricate una sola volta nella memoria (scenario molto frequente con i loop)
  - Accopiato al pre-fetch
- Predizione dei salti:
  Si possono avere due tipi di approcci:
  - **Statico** ,
    prevedo di saltare *sempre* ,
    prevedo di *non* saltare *mai* ,
    prevedo di saltare in base al *codice operativo*
  - **dinamico** ,
    bit take/not taken,
    tabella dell storia dei salti,
    Questo approccio cerca di migliorare la qualità della predizione del salto memorizzando la *cronologia delle istruzioni di salto condizionato* di un certo programma.

Per questo ad **ogni** istruzione di salto condizionato associo **1 o 2 bit** per ricordae l'andamento delle ultime istruzioni. I bit vengono memorizzati in una locazione temporanea ad accesso **molto veloce** .

* 1 bit,
  ricorda come è andata l'ultima volta quindi predice di comportarsi in maniera uguale:
  se **1** predico di saltare,
  se **0** predico di non saltare,
  se **sbaglio** predizione inverto il bit.
* 2 bit,
  Ricorda come è andata la predizione degli ultimi due salti, per invertire la predizione si ha bisogno di due errori consecutivi

- Salto ritardato,
  finchè non si è a conoscenza dell'esito dell'istruzione di salto, al posto di rimanere in stallo si esegue un'istruzione che non dipende dal salto, il compilatore alloca un'istruzione 'opportuna' subito dopo l'istruzione di salto quindi la CPU esegue sempre PRIMA l'istruzione presente nel *branch delay slot* e dopo altera l'ordine di esecuzione delle istruzioni.

2.4 CISC e RISC

Nell'evoluzione dei calcolatori si investono molti più capitali nel reparto **software** piuttosto che in quello **hardware**. Con l'avvenire dei *linguaggi ad alto livello* (HLL è più semplice esprimere algoritmi complessi in maniera coincisa e delegano al compilatore il compito di tradurre questi in *linguaggio macchina* . Questi HLL supportano costrutti di programmazione strutturata, ovvero i diversi paradigmi.
Per poter ridurre il *gap semantico* ovvero ciò che sta fra le istruzioni in HLL a quelle in linguaggio macchina, una soluzione dei progettisti hardware è stata quella di:

- Ampliare il set delle istruzioni
- Aggiunta di diversi modi di indirizzamento
- Implementazione hardware di costrutti di linguaggi ad alto livello

Così si semplifica il lavoro del compilatore, l'esecuzione diventa più efficiente, si possono supportare HLL più complessi.
Un approccio diverso a questa soluzione può essere:

- Individuazione delle caratteristiche e e dei pattern di esecuzione delle istruzioni macchina generate dai programmi in HLL
- Semplificazione dell'architettura di base piuttosto che la sua complicazione

2.4.1 RISC

Il processo di semplificazione coinvolge:

- Le operazioni eseguite,
  semplificare le funzionalità del processore e la sua interazione con la memoria

- operandi,
  tipo e frequenza d'uso degli operandi sono alla base dell'organizzazione della memoria e dei modi di indirizzamento

- Flusso dell'esecuzione,
  organizzazione della pipeline e del controllo

Per rendere possibile questa semplificazione è necessario analizzare le istruzioni macchina generate dai programmi HLL, e analizzare le misure dinamiche che si raccolgono con l'esecuzione del programma e contano il numero di occorrenze di una certa proprietà o di una certa caratteristica.
Per esempio nei linguaggi ad alto livello si hanno molte occorenze delle istruzioni di assegnamento e delle istruzioni condizionali. Oltre alla frequenza di un'istruzione è importante tenere in conto il tempo d'esecuzione di un'istruzione.

Esito Ricerca Il risultato della ricerca eseguita in particolare Hennessy e Patterson negli anni '80 porta alla conclusione che la strategia migliore per supportare i linguaggi ad alto livello è di:

- **NON** rendere le istruzioni macchina simili a quella di HLL
- **OTTIMIZZAZIONE** delle performance e dei pattern più usati e più time-consuming
- **ampio numero di registri** e il compilatore li utilizza in maniera ottimizzzata
- **pipeline** accuratamente progettata
- **set di istruzioni semplificato, RISC** ed efficientemente implementato

Uso dei regsitri Un registro è una memoria interna alla CPU ad accesso molto rapido (+ veloce della cache), quindi hanno indirizzi più brevi di quelli della cache e della memoria principale, è importante che gli *operandi* utilizzati siano conservati più a lungo nei registri cosicchè da ridurre i trasferimenti memoria-registro.

- **Hardware** ,
  aumenta il numero di registri,
  più variabili sono mantenute per più tempo

- **Software** ,
  il compilatore massimizza l'uso dei regsitri,
  le variabili più usate per ogni intervallo di tempo sono allocate nei registri,
  richiede una sofisticata analisi dell'utilizzo dei programmi

Per poter utilizzare i molti registri general purpose un'idea è quella di suddividere i registri in molti piccoli gruppi (finestre) e per ogni procedura viene utilizzata un gruppo diverso.
Ogni gruppo è composto da:

- Registro parametri,
  contiene i parametri che vengono passati quando la procedura viene chiamata e contiene il valore da restituire al chiamante al termine della procedura

- Registri locali,
  memorizza il contenuto delle variabili locali

- Registri temporanei,
  Scambia parametri e il valore di ritorno con un'eventuale procedura chiamata,
  molto spesso il registro temporaneo di una procedura i è il registro dei parametri della procedura successiva $i + 1$ a livello fisico, quindi non c'è nemmeno il bisogno di trasferire i dati

Buffer Circolare I registri sono spesso organizzati in un regitro circolare, in esso è presente il **CWP** , current window pointer, come suggerisce il nome esso indica la finestra corrente, mentre il **SWP** , saved window pointer salva l'indrizzo della finestra in cui si deve ritornare al termine della procedura in corso.

Variabili Globali esse sono variabili che sono accessibili da qualunque procedura e da più di esse. Il compilatore le allocca in memoria, questa cosa è però poco efficiente in quanto vengono usate spesso e l'utilizzo della memoria è un'operazione molto onerosa. La **soluzione** a questo problema è quello di usare un **gruppo di registri ad hoc** che sono disponibili a tutte le procedure.

Otimizzazione dei registri Lo scopo di quest'operazione è quella di trovare gli operandi il più possibile nei registri e minimizzare le operazioni di *load/store* . L'implementazione *software* avviene attraverso l'ottimizzazione del compilatore e mediante l'utilizzo di **registri simbolici** .
Ogni variabile viene mappata ad un registro reale e *se* due variabili vengono utilizzate in momenti diversi possono essere mappate sullo *stesso registro* , il che può rendere il numero dei regsitri quasi infinito.
Se il numero dei registri non è sufficiente per contenere *tutte* le variabili esse vengono memorizzate nella *memoria principale* .
Il numero di registri che vengono generalmente utilizzati nell'architettura *RISC* sono fra i 32 e i 64.

2.4.2 CISC

Il tipo di architettura **CISC** *Complex Instruction Set Computer* a differenza dell'architettura RISC ha un'*ampio* insieme di istruzioni, e ha istruzioni **più** complesse al fine di *semplificare* il lavoro del compilatore e migliorarne le performance.
Tuttavia, non semplifica le istruzioni ma le rende più complesse e più simili ai linguaggi ad alto livello, inoltre con set di istruzioni più complesso è difficile *ottimizzare il codice macchina* per ridurlo e riorganizzarlo.
Le istruzioni più complesse possono essere eseguite più *velocemente* **ma** al costo di

- unità di controllo più complessa
- controllo microprogrammato necessita più spazio
- rallentamento dell'esecuzione delle istruioni più semplici, che rimangono *le più frequenti*

2.4.3 Confronto fra CISC e RISC

Istruzinoi per ciclo di clock

RISC : il ciclo esecutivo di quest'architettura dura un solo machine cycle, *se la pipeline è piena* si termina un'istruione ad **ogni** ciclo di clock

CISC : le istruzioni di questo tipo impiegano più di un ciclo di clock per essere terminate.

Memorie usate per le operazioni

RISC : sono tutte fra registri tranne che per la **load/store** che sono fra registri e memoria.

CISC : hanno anche operazioni *memory-memory* e *register-memory* . Dato che vengono spesso utilizzati *scalari locali* aumentando e/o otttimizzando i registri si ha un incremento delle prestazioni.

Modi di indirizzamento

RISC : si usano pochi e semplici modi di indirizzamento (si semplifica l'istruzione)
CISC : si utilizzano molti modi di indirizzamento e alcuni complessi

Caratteristiche architetturali

RISC: pochi e semplici formati *fissi* per le istruzioni, decodifica opcode e accesso ai registri può essere simultaneo

Verdetto

Tuttavia non si può notare quale architettura sia la *migliore* ma ciò dipende dai campi applicativi in cui queste vengnono utilizzate, è difficile capire quanta influenza viene data dal *compilatore* , (per esempio un buon compilatore CISC può rendere quest'architettura migliore nel confronto se si ha uno scarso compilatore RISC). Inoltre, la maggior parte dei confronti eseguiti sono stati fatti su prototipi semplificati e non su macchine ad uso commerciale. Al giorno d'oggi alcune architetture utilizzano aspetti caratteristici delle altre, per esempio architetture RISC con elementi CISC o viceversa.
Altre componenti cominciano a dover essere tenute in considerazione come:

- GPU, graphic processing unit
- TPU, tensore processing unit, reti neurali e processori per machine learning

2.4.4 Banco registri vs Cache

- Banco Registri,
  - contiene **tutti** gli scalari locali
  - variabili individuabili
  - variabili globabli assegnate a registri specifici
  - save/restore basato sulla profondità di annidamento
  - indirizzamento a registro
- Cache,
  - *solo* gli scalari usati di recente
  - blocchi di memoria
  - *solo* le variabili globali usate di recente
  - save/restore basato sull'algoritmo di sostituzione della cache

– indirizzamento a memoria

Per riferirsi agli scalari locali i registri sono più veloci della cache.

2.5 MIPS

2.6 Processori Multi-Core