La macchina di Von Neumann:

Per superare le difficoltà della programmazione cablata, con la macchina di Von Neumann vennero introdotti dei nuovi concetti:

- memoria centrale contiene dati e istruzioni accessibili per indirizzo
- l'esecuzione delle istruzioni avviene in modo sequenziale
- Alu in grado di operare su dati binari
- unità di controllo che interpreta istruzioni in memoria e le manda in esecuzione
- dispositivi di I/ azionati dal unità di controllo

Linee generali:

- il calcolatore in quanto tale dovrà contenere un unità aritmetica in grado di eseguire operazioni sui numeri
- il controllo delle operazioni viene affidato ad un organo centrale
- Il dispositivo deve possedere più memoria possibile per svolgere operazioni complesse
- devono esserci dei circuiti dedicati agli ingressi ed alle uscite di dati tra i moduli del calcolatore, da R a M in ingresso e da M a R

Registri:

- ***MBR Memory Buffer Register:*** contiene la parola da scrivere in memoria
- ***MAR Memori Address Register:*** contiene l'indirizzo della parola di memoria in cui scrivere il MBR
- ***IR Instruction Register:*** contiene i bit del codice operativo dell'istruzione corrente (Opcode)
- ***PC Program Counter:*** contiene l'indirizzo della prossima istruzione da eseguire
- ***AC Accumulator:*** contiene temporaneamente gli operandi e risultati parziali delle operazioni dell'Alu.

Ciclo esecutivo:

Alcuni stadi implicano uno scambio dati tra processore e memoria, altri riguardano il solo processore:

- ***Calcolo indirizzo istruzione CII:*** determina l'indirizzo della successiva istruzione da eseguire
- ***Fetch F:*** legge l'istruzione dalla memoria e la trasferisce nel processore
- ***Decodifica Istruzione DI:*** analizza l'istruzione per determinare gli operandi e il tipo da usare
- ***Calcolo indirizzo operando CIO:*** determina l'indirizzo dell'operando se l'operazione implica un riferimento a un operando in memoria oppure ottenibile da una periferica.
- ***Lettura operando LO:*** legge l'operando dalla memoria o periferica
- ***Operazione su dati OD:*** esegue l'operazione indicata nell'istruzione
- ***Memorizzazione operando MO:*** scrive il risultato in memoria o lo invia ad una periferica
- ***Controllo interrupt ed eventuale gestione***

Interrupt:

Le operazioni di i/o sono molto lente quindi se il processore dovesse bloccare il proprio lavoro per gestirle una alla volta si verificherebbe un netto calo di prestazioni.
Per questo motivo gli interrupt consentono ai moduli di i/o di inviare delle richieste di interrupt alla cpu che, una volta gestita ed avviata l'istanza, riprende il proprio lavoro in attesa di un interruzione successiva.
Il tutto viene gestito da processore e sistema operativo: il programma utente non deve contenere nessun paramento per gestire le interruzioni.

Al ciclo esecutivo si aggiunge un controllo d interrupt, se vi sono interrupt pendenti la CPU salva in contesto corrente, carica nel PC la prima istruzione del interrupt handler o routine di gestione(di solito fa parte del sistema operativo).
In questa fase viene determinata la natura dell'interrupt e le azioni da svolgere, per questo motivo si genere del Overhead.
In caso di interrupt multipli vi sono due approcci possibili:

- ***interrupt disabilitato:*** la cpu ignora la richiesta che resterà pendente e verrà servita solo al termine del interruzione corrente.
- ***Interrupt a priorità:*** viene tenuto conto della priorità della richiesta, infatti una linea dati avrà la precedenza sulla stampa. In questo sistema le interruzioni vengono "annidate".

In caso di trasferimenti da i/o a memoria viene interpellato il DMA che bypassa l'utilizzo della cpu.

Classi di Interrupt:

- ***Programma****:* generato da una condizione derivate dall'esecuzione di un istruzione (overflow, divisione per zero…)
- ***Timer:*** generato da un timer del processore
- ***I/O:*** gen da un controllore I/O es completamento di un operazione o errore
- ***Guasto Hardware:*** errore parità, mancanza alimentazione

Bus:

Un bus è un canale di comunicazione che connette le componenti del calcolatore, esso è composta da più linee che trasmettono in parallelo un dato, il numero di linee determina quanti bit posso essere trasportati contemporaneamente, per trasferire 8 bit in un solo ciclo clock ci vorranno 8 linee.
Bus con un gran numero di dispositivi collegati soffrono di ritardo di propagazione dovuto al tempo dedicato al coordinamento di tutte le periferiche, inoltre quando la domanda di trasferimento si avvicina alla capacità del bus si verifica un collo di bottiglia, risolvibile aumentando le velocità e usando bus dedicati.
Il bus di espansione bufferizza i trasferimenti di dati tra il bus di sistema e i controllori di i/o sul bus di espansione, in questo modo è possibile connettere un gran numero di dispositivi (fax, 56k, seriale, ecc.) isolando allo stesso tempo il traffico dalla memoria al processore dal traffico di i/o.

Tipi di linee:

- ***dedicate:*** linee separate per indirizzi e dati, ha il vantaggio fi avere un alto throughput poiché vi è meno contesa del bus ma ha costi e dimensioni maggiori.
- ***multiplexate:*** in un'unica linea vengono trasmesse le informazioni di indirizzo e dati; utilizzando una linea di controllo Address Valid si segnala il transito di informazioni di controllo o dati, ogni modulo può così copiare l'indirizzo e capire se è il destinatario dei dati che varranno trasmessi. Questa tecnica ha il vantaggio di essere economica e poco onerosa in termini di spazio ma richiede circuiti più complessi e le prestazioni sono limitate.
-

L'utilizzo del bus può essere gestito in modo centralizzato o distribuito; nel prmo vi sarà un controller che decide a chi assegnare il bus, nel secondo sono i moduli stessi ad autogestirsi utilizzando speciali algoritmi.
Vi sarà così un master che tramette e uno slave che riceve.

La temporizzazione del bus può essere sincrona, dove ogni occorrenza degli eventi è determinata da un clock, o asincrona, dove l'occorrenza di un evento sul bus dipende dall'occorrenza di un evento precedente.

Tipi di Bus:

- ***Linee dati:*** sono il percorso su cui viaggiano i dati, il numero di linee viene detto ampiezza ed è un parametro fondamentale per le prestazioni del calcolatore,
- ***Linee indirizzi:*** determinano la destinazione dei dati, l'ampiezza del bus indirizzi stabilisce la quantità di memoria massima; in genere i bit più significativi indicano la periferica e quelli meno significativi indicano una locazione di memoria o una porta i/o
- ***Linee di controllo:*** controllano l'accesso e l'uso delle linee dati e indirizzi. Comprendono segnali di temporizzazione e comandi.
- ***Linee per alimentazione elettrica***

Memorie:

La "parola" è l'unità naturale di organizzazione della memorie e la sua dimensione corrisponde al numeri di bit utilizzati per rappresentare un numero e alla dimensione delle istruzioni.
La soluzione ai problemi legati alle memorie risiede nel non fare affidamento su un unico componente bensì adottare una gerarchia di memoria dove le memorie più piccole, più costose e veloci sono integrate da memoria più grandi, economiche e lente.
Il principio della località dei riferimenti ci insegna che durante l'esecuzione di un programma i riferimenti alla memoria da parte del processore tendono a raggrupparsi. Sfruttando questo principio possibile diminuire la frequenza di accessi da parte del processore, è infatti possibile organizzare i dati nella gerarchia in modo che la percentuale di accessi a ciascun livello immediatamente più basso risulti sostanzialmente minore di quella al livello sovrastante.

Metodi di accesso ai dati:

- ***sequenziale:*** la memoria è organizzata in record e l'accesso deve avvenire in uno specifico ordine (nastro)
- ***diretto:*** unica unità di letture/scrittura che accede a singoli blocchi rappresentati da un indirizzo univoco , implica una ricerca sequenziale o un attesa ed un posizionamento (dischi)
- ***casuale:*** ogni locazione indirizzabile ha il proprio circuito di indirizzamento, il tempo di accesso è costante (ram)
- ***associativo:*** una certa configurazione di bit viene confrontata simultaneamente con gli indirizzi di tutte le locazioni (cache)

Prestazioni:

- ***tempo d'accesso/latenza:*** il tempo che intercorre tra l'istante in cui l'indirizzo viene presentato alla memorie e l'istante in cu i dati sono stati memorizzati o resi disponibili per l'uso. Negli HDD è il tempo di posizionamento della testina.
- ***Durata del ciclo di memoria:*** tempo di accesso più qualsiasi tempo addizionale richiesto prima che possa cominciare un secondo accesso.
- ***Tasso di trasferimento:*** è la frequenza alla quale i dati possono essere trasferiti nella o dalla unità di memoria. Per la memoria ad accesso non casuale vale Tn=Ta + (N/R) dove Tn tempo medio per leggere o scrivere N bit, Ta tempo medio di accesso, N numero di bit, R tasso di trasferimento in bps.

Cache:

La cache contiene una copia di parti della memoria centrale; quando il processore tenta una lettura in memoria centrale viene controllato se sia già presente in cache, in caso negativo si porta un blocco (numero prefissato di parole) in cache e viene fornita alla cpu la parola richiesta. Sfruttando il principio della località dei riferimenti, le successive richieste di parole dovrebbero riguardare quelle precedentemente copiate.
Poiché ci sono più blocchi che linee cache bisognerà adottare delle politiche di trasferimento opportune per garantire una probabilità di cache hit maggiore del 90%. Per ottenere ciò si è visto che aumentare troppo la dimensione della cache non aiuta in quanto cresce il numero di circuiti di indirizzamento.

Tipi di mapping:

- ***diretto:*** assegna a ciascun blocco di memoria una sola possibile linea di cache. Campi: Tag, Linea, Parola. Tecnica semplice e poco costosa, se un programma accede ripetutamente a parole di due blocchi a cui è assegnata la stessa linea vi è un alto numero di miss.
- ***associativo:*** i blocchi di memoria possono essere caricati in qualsiasi linea di cache. Campi: tag, parola. Lo svantaggio è la complessità circuitale richiesta per esaminare in parallelo i tag di tutte le linee di cache.
- ***Set-associativo:*** la cache è suddivisa in insiemi (set) di k linee. Il blocco B può essere assegnato a qualunque linea dell'insieme i. Campi: tag, insieme/set, parola.

Algoritmi di sostituzione:

- ***LRU:*** sostituisce il blocco dell'insieme che è rimasto nella cache più a lungo senza essere referenziato, ogni linea include un bit USE, la linea usata setta il bit a 1 mentre le altre linee dell'insieme settano 0, quando un blocco viene trasferito i ncache andrà a sostituire la linea in cui il bit USE è a 0.
- ***FIFO:*** sostituisce il blocco nell'insieme che è rimasto più a lungo nella cache.
- ***LFU:*** sostituisce il blocco dell'insieme che ha subito meno accessi, si associa un contatore a ciascuna linea.
- ***Casuale:*** delle simulazioni hanno mostrato che le prestazioni di questa politica sono appena inferiori rispetto agli altri algoritmi.

Politiche di scrittura:

I dati presenti nella gerarchia di memoria possono essere modificati da componenti diversi (cpu-> cache, i/o -> memoria centrale) vi è quindi la necessità di assicurarne l'effettiva consistenza, per questo si usano le politiche di scrittura:

- ***write-through:*** tutte le operazioni vengono eseguite sia in memoria centrale si in cache, traffico notevole
- ***write-back:*** gli aggiornamenti vengono eseguiti sono nella cache; ogni linea cache contiene un bit UPDATE (dirty bit) inizializzato a 0 e viene posto a 1 durante la prima scrittura nella linea, quando una linea deve essere sovrascritta viene copiata in memoria centrarle solo se il sua UPDATE è 1. Gli accessi tramite i/o passano per la cache e questo richiede circuiti complicati; vi è un ulteriore problema in caso di più cache ed una memoria condivisa tra dispositivi.
    - ***monitoraggio del bus con write-trough:*** controllori delle cache intercettano le modifiche alle locazioni condivise
    - ***trasparenza hardware:*** circuiti aggiuntivi per assicurare che tutte le modifiche in memoria centrare si riflettano in tutte le cache

- ***memoria non-cachable:*** solo una parte di memoria viene condivisa e non cachable

Tipi di miss:

- ***primo accesso:*** inevitabile
- ***capacità insufficiente:*** quando la cache non può contenere ulteriori blocchi
- ***conflitto:*** quando più blocchi possono essere mappati sullo stesso gruppo

Soluzioni ai problemi di miss:

- *dimensione blocco maggiore:* sfrutta località dei riferimenti ma aumenta miss per conflitto (meno blocchi disponibili).
- *Maggiore associatività:* incrementa tempo di localizzazione, regola 2:1 una cache ad N blocchi con associazione diretta ha una probabilità di miss uguale ad una cache di dimensione N/2 con associazione a 2 vie.
- *Cache multilivello*
- *Cache separata tra dati e istruzioni*
- *Ottimizzazione tramite compilatore (cicli)*

Memorie a semiconduttore:

La memoria si basa sulla cella su cui è possibile leggere, scrivere e può assumere due stati stabili (0/1). Ogni cella presenta tre porte: selezione, controllo e lettura/scrittura.
Le memorie RAM sono ad accesso casuale che permettono una facile lettura e scrittura di dati tramite segnali elettrici, sono volatili cioè devono disporre di alimentazione costante per non perdere quanto memorizzato, si suddividono in DRAM e SRAM.

DRAM vs SRAM:

La DRAM è una ram dinamica, i dati sono segnali elettrici memorizzati in condensatori e la presenza o assenza di carica confrontata con un valore di soglia viene interpretata come bit 1 o 0. Data la natura del condensatore a scaricarsi c'è bisogno di circuiti che periodicamente effettuino un refresh, inoltre l'operazione di lettura scarica il condensatore che andrà opportunamente ricaricato.
La SRAM è costituita da quattro dispositivi digitali quali i transistor che memorizzano valori binari senza la necessità di refresh.
LA ram dinamica è più piccola ed economica quindi più densa ma necessitano di hardware per il refresh, sono indicate per grandi quantitativi di memoria. Le ram statiche sono invece molto costose e veloci, che vengono utilizzate in piccoli quantitativi come nella cache.

ROM:

Memorie in sola lettura non sono volatili ma non è possibile scrivere nuovi dati. Vengono utilizzate nella microprogrammazione per memorizzare subroutine, programmi di sistema e funzioni tabulate; per la produzione vi è un costo fisso per lo stampaggio litografico della matrice che viene ammortizzato con l'altro numero di unità prodotte. Esistono delle PROM o rom programmabili scrivibili elettricamente una sola volta con apparecchiature apposite. EPROM permettono cancellazioni ottiche tramite UV e riscritture elettroniche. EEPROM permettono cancellazione e scrittura elettronica ma richiedendo la scrittura tempi proporzionalmente grandi rispetto alla lettura sono indicate per riscritture occasionali. Nelle memorie Flash è possibile cancellare selezioni di celle in un'unica azione.

Correzione degli errori:

Tutti i tipi di memorie sono soggetti a errori che possono essere hardware (guasti) o software (evento casuale che altera il contenuto di alcune celle dovuti a problemi di alimentazione o particelle Alpha). Per far fronte a questi problemi le memorie implementano un sistema di controllo e correzione; prima di memorizzare i dati si esegue un calcolo per produrre un codice che verrà memorizzato insieme al dato. La dimensione effettiva della parola sarà data da dato+codice. Quando si legge una parola si genere nuovamente un codice che viene confrontato con quello precedentemente memorizzato, se viene rilevato un errore si prova a correggerlo tramite il correttore. Il codice di correzione di Hamming raggruppa i bit in insiemi, utilizza dei bit di parità in modo che il numero totale di 1 in ciascun insieme sia pari. Se un errore modifica uno dei bit dati verrà riconosciuto controllando i bit di parità.
I circuiti di confronto ricevono in ingresso due stringhe di K bit di controllo (una letto dalla memoria, l'altra generata dal dato letto dalla memoria), il confronto bit a bit viene effettuato eseguendo lo XOR dei due ingressi generando così la parola sindrome. La sindrome, ampia K bit, ha valore da 0 a (2^K)-1 e indica quale bit sia errato. (2^K)-1 >= M + K. S la sindrome contiene tutti 0 non vi è errore, se contiene un unico bit a 1 vi è un errore nei bit di controllo, se contiene più di un bit a 1 allora il valore numerico della sindrome indica la posizione le bit errato che andrà complimentato. Per ottenere questo i bit di dati e controllo sono distribuiti nella sindrome in modo che le posizioni il cui numero è una potenza di 2 siano assegnate ai bit di controllo. Ogni bit di controllo opera sui bit del dato il cui numero di posizione contiene un 1 nella stessa posizione del numero di posizione di quel bit di controllo.
Il codice di Hamming viene detto a correzione di singolo errore, in genere le memorie moderne sono dotate di rilevazione doppia di errore, necessitano di un ulteriore bit di parità impostato in modo che il numero totale di 1 nel diagramma sia pari.

Dischi magnetici:

Il disco magnetico è composto da un piatto di vetro rotante chiamato substrato rivestito da un materiale magnetizzabile. La lettura e scrittura vengono effettuate da un braccio meccanico mobile che monta due apposite testine; quando lo strato magnetizzabile passa sotto la testina di scrittura basterà polarizzare quest'ultima in uno dei due versi per trasferire la carica al disco, la lettura avviene in modo analogo essendo il disco polarizzato e la testina in grado leggerne la polarità.
I dati sono disposti in anelli concentrici chiamati tracce separate da spazi per prevenire interferenze. Ogni traccia è composta da molti settori che costituiscono l'unità minima di lettura. Il disco viene inizializzato con dati invisibili all'utente che permettono al sistema di controllo di gestire il posizionamento della testina e la sincronizzazione. L'insieme delle tracce nella stessa posizione relativa sul piatto viene detto cilindro. La testina del disco Winchester , sfruttando la pressione d'aria generata dalla rotazione del disco, riesce ad operare più vicino alla superfice del disco permettendo una densità di dati maggiore.

Prestazioni HDD:

- ***posizionamento/seek:*** tempo richiesto per posizionare la testina sulla traccia.
- ***Ritardo/latenza:*** tempo di attesa in cui il settore si posiziona sotto la testina
- ***Tempo di accesso:*** seek + latenza
- ***Tempo di trasferimento:*** frazione di tempo in qui il settore si muove sotto la testina, avviene il trasferimento dei dati. T = b / (r*N) dove b byte da trasferire, r rotazioni al secondo, N byte per traccia.

Vanno considerati in aggiunta i tempi associati all'operazione di i/o come l'effettiva disponibilità della periferica e dei canali di trasferimento.

RAID:

Viste le difficoltà ad aumentare le prestazioni dei dischi fissi sono state sviluppati degli array di dischi che operano in parallelo; utilizzando più dischi si hanno svariati modi per organizzare i dati e sarà possibile aggiungere delle informazioni ridondanti.
Lo schema RAID (redundant array of indenpendet disks) fornisce 7 diverse configurazioni con le seguenti caratteristiche comuni:

- L'insieme di dischi fisici viene visto dal sistema operativo come una singola unità logica
- i dati sono distribuiti sui dischi
- la capacità ridondante viene utilizzata per memorizzare informazione di parità

nota: i sistemi RAID 2, 4 non sono mai stati commercializzati.

- ***0:*** il raid 0 non include la ridondanza dei dati a favore delle prestazioni, i dati sono distribuiti sui vari dischi quindi se vi sono due richieste di blocchi che si trovano in dischi diversi esse vengono servite in parallelo. I dati vengono distribuiti in strisce distribuite a rotazione (round robin) sui dischi.
- ***1:*** in raid 1 la ridondanza viene ottenuta duplicando tutti i dati, questo implica un numero doppio di dischi fissi. Si usa lo striping dei dati e ogni striscia si trova fisicamente su due dischi quindi una lettura può essere soddisfatta dal disco che si accede più velocemente al dato richiesto, la scrittura invece viene eseguita da entrambi i dischi quindi le prestazioni sono condizionate dalla più lenta delle scritture. In caso di guasto i dati sono disponibili nell'altro disco, questo permette un alta affidabilità ma un costo elevato
- ***2:*** (non commercializzato), sfrutta l'accesso parallelo ossia le testine di ciascun disco si torva nella stessa posizione in ogni momento. Si uso lo striping dei dati con strisce molto piccole (byte o parola), vengono generati i codici di correzione errori di Hamming memorizzati in più dischi, sistema dispendioso richiede molti dischi fissi.
- ***3:*** usa un solo disco ridondante che memorizza i bit di parità per ogni striscia nei dischi dati. In caso di guasto di un disco i dati mancanti vengono generati al volo (reduced mode) sfruttando i bit di parità. Ottime prestazioni ma vi è un overhead per il calcolo del bit di parità.
- ***4:*** (non commercializzato) ogni disco opera indipendentemente quindi letture molto veloci, scrittura lenta a causa del calcolo della parità, ottimo per grandi richieste di i/o, usa lo striping dei dati con strisce grandi; parità bit a bit calcolata tra unità di informazione per ogni disco, le informazioni di parità vengono memorizzate su un disco ad hoc.
- ***5:*** simile al 4 ma distribuisce le strisce di parità su tutti i dischi seguendo lo schema a rotazione (round robin) evita il collo di bottiglia del disco di parità del raid 4. Usato nei server di rete
- ***6:*** si effettuano due calcoli di parità che sono memorizzati in blocchi separati su dischi differenti, necessari n+2 dischi di quelli richiesti, assicura un alta affidabilità (devono guastarsi tre dischi), scrittura lenta.

Memorie ottiche:

Il CD è un dispositivo ottico non cancellabile che sfrutta una serie di pozzetti (pit) per memorizzare dati binari su una superfice di policarbonato. Un laser è in grado di distinguere i pit dai land si distingue così il segnale digitale. La traccia forma una spirale che inizia vicino al centro e termina sul bordo del disco in modo che i settori mantengono una lunghezza costante.

Input/Output:

I moduli di i/o operano per evitare problemi legati alle diverse logiche di funzionamento delle periferiche, per compensare le differenze di velocità di funzionamento e per tradurre differenti formati di dati. Esistono tre categorie di dispositivi esterni:

- ***comprensibili dall'uomo:*** utente – calcolatore (video, tastiera..)
- ***comprensibili dalla macchina:*** comunicazione tra apparecchiature (controllo, monitoraggio)
- ***comunicazione:*** per comunicare con dispositivi remoti (modem, rete)

L'interfaccia verso il modulo di i/o è composta da segnali di controllo, di dati e di stato.

- ***segnali di controllo****:* determinano la funzione che il dispositivo eseguirà
- ***segnali di dati:*** insiemi di bit da scambiare
- ***segnali di stato:*** indicano lo stato del dispositivo (pronto/non pronto)

La logica di controllo controlla le operazioni su comando del modulo. Il trasduttore converte i dati dalla forma elettrica ad altere forme d'energia.

Moduli di I/O:

Le funzioni del modulo di i/o sono: controllo & temporizzazione, comunicazione con Cpu, comunicazione con i dispositivi, buffering dei dati, rilevazione degli errori. Un esempio di trasferimento dati da un dispositivo esterno al processore implica i seguenti passi:

- cpu interroga il modulo i/o sullo stato del dispositivo connesso
- il modulo i/o restituisce lo stato del dispositivo
- se il dispositivo è pronto a trasmettere, la cpu richiede il trasferimento dei dati tramite un comando rivolto al modulo i/o
- il modulo i/o ottiene un'unità di dati dal dispositivo esterno
- i dati vengono trasferiti dal modulo i/o al processore

Comandi di I/O:

- ***controllo:*** attiva una periferica e comunica cosa fare
- ***test:*** testa le condizioni di stato dei moduli i/o e delle periferiche
- ***lettura:*** ottiene dati dalla periferica
- ***scrittura:*** il modulo scrive sula periferica i dati presenti nel buffer

Istruzioni di I/O:

I moduli interpretano il bus degli indirizzi per determinare a chi è rivolto il comando, in caso di bus condiviso vi sono due modalità di indirizzamento:

- ***memory mapped:*** non c'è distinzione tra spazio indirizzi di locazioni di memoria e dispositivi, le istruzioni macchina usate per accedervi sono le medesime
- ***separato:*** il bus è dotato di una linea di selezione aggiuntiva che specifica se l'indirizzo è rivolto ad una periferica o alla memoria, in questo modo raddoppiano il numero totale di indirizzi disponibili; comandi speciali per i/o

Tecniche di gestione I/O:

- ***I/O da programma:*** la Cpu controlla direttamente il modulo i/o (stato, comandi, trasferimenti) invia i comandi e controlla periodicamente lo stato del modulo. Il processore attende un lungo periodo prima che il modulo sia pronto e deve eseguire ripetuti controllo sullo stato di quest'ultimo perdendo così del tempo.
- ***I/O interrupt driven:*** il processore invia un comando di i/o al modulo e prosegue nel suo lavoro finché il non riceverà un interrupt. Questo metodo evita l'attesa da parte

della cpu. Se è presente un interruzione viene salvato il contesto corrente, PC e registri, e viene elaborato l'interrupt.

- ***DMA:*** è un modulo addizionale sul bus di sistema e interviene nei trasferimenti tra moduli e memoria. Se necessario il DMA è in grado di sottrarre cicli di bus al processore (cycle stealing). Se il processore deve leggere/scrivere attraverso il DMA invia le seguenti informazioni:
    - Se si tratta di lettura/scrittura
    - L'indirizzo del dispositivo di I/O
    - La locazione inziale in memoria che verrà memorizzata dal DMA nel proprio registro indirizzo
    - Il numero di parole da leggere o scrivere

Il processore prosegue con altro lavoro e riceve un interrupt solo quando viene completato il trasferimento.
Il controller DMA può accedere al canale in due modi:

- ***una parola alla volta (cycle stealing):*** sottrae di tanto in tanto alla cpu il controllo sul canale
- ***per blocchi (burst mode):*** prendendo possesso del canale per una serie di trasferimenti, più efficace poiché l'acquisizione del canale è onerosa.

Aritmetica del calcolatore:

L'Alu è l'unità che si occupa di effettuare i calcoli sui dati forniti dal resto del calcolatore, essa genera risultati destinati ai registri e può impostare flag (es. overflow).

Rappresentazione in modulo e segno:

Il bit più significativo rappresenta il segno (1 se -), i restanti numeri rappresentano il modulo. Gli svantaggi di questa rappresentazione sono che vanno sempre considerati i segni dei due numeri e i due moduli, inoltre lo zero ha ben due rappresentazioni. Per estendere il segno basta spostare il bit di segno nella cifra più a sinistra e riempire gli spazi restanti di zeri.

Complemento a due:

Il bit più significativo indica il segno.
Per n bit è possibile rappresentare da -2^(n-1) a +2(n-1)-1. I numeri positivi si rappresentano con il primo bit 0 e gli altri in normale forma binaria. Per i negativi si esegue il complemento bit a bit del numero positivo corrispondente e si somma 1 al risultato considerandolo un intero senza segno. L'estensione del segno avviene aggiungendo nuovi bit a sinistra uguali al bit di segno originale. Il complemento a due ha il vantaggio di avere una sola rappresentazione dello zero e semplifica le operazioni aritmetiche (compreso l'opposto).

- ***Opposto:*** eseguire in complemento bit a bit (incluso il bit di segno), trattare il risultato come intero binario senza segno e sommare 1.
- ***Somma:*** se si ottiene un numero positivo esso è in notazione binaria naturale; se si ottiene un negativo si ottiene un numero in complemento a due. L'overflow viene segnalato quando la somma di due numeri di segno uguale genera un risultato di segno opposto.
- ***Sottrazione:*** per sottrarre un numero si considera l'opposto del sottraendo e lo si somma al minuendo.
- ***Moltiplicazione:*** molto complessa, va calcolato i prodotto parziale per ogni cifra e poi vanno sommati. Genera un risultato di 2n bit. Viene utilizzato l'algoritmo di Booth.
- ***Divisione:*** più complessa della moltiplicazione ma si basa sugli stessi principi, usa traslazioni, somme e sottrazioni ripetute.

Rappresentazione in virgola mobile:

Utilizzando una stringa suddivisa nei campi segno, significando ed esponete è possibile rappresentare un gran numero di cifre comprendendo anche le posizioni decimali. La base è sottintesa, si assume che la virgola si trovi a destra della cifra più a sinistra del significando. Il bit più a sinistra del numero in virgola mobile è il segno, l'esponente si trova nei successivi 8 bit; viene usata a rappresentazione polarizzata (biased), un valore fisso viene sottratto dal campo per ottenere il vero valore dell'esponente, in genere vale 2^(k-1)-1 con k numero di bit dell'esponente, questo permette di confrontare i numeri in virgola mobile non negativi come interi. Il significando, o mantissa, è un numero normalizzato cioè ha la cifra più significativa a 1 e poiché questo bit è sempre 1 non è necessario memorizzarlo (1,b..b x esp).
Con 32 bit si rappresentano sempre 2^32 numero in totale ma vengono distribuiti in intervalli non equidistanti positivi e negativi, in prossimità dell'origine questi spazi si addensano; il solo modo per incrementare sia l'intervallo sia la precisone è utilizzare più bit.
In questa rappresentazione non è previsto un valore per lo 0 anche se speciali schemi di bit designano dei casi particolari.

IEEE 754:

Questo standard viene utilizzato da tutti i processori ed è stato sviluppato per facilitare la portabilità di programmi aritmetici complessi, definisce un formato a 32bit e uno doppio a 64bit con esponenti da 8 e 11bit. Esistono anche due formati estesi singolo e doppio che includono bit aggiuntivi nell'esponente e nel significando. I valori estremi per l'sponete (tutti 0 o tutti 1) definiscono valori speciali. Esponente 0 con frazione di 0 rappresenta lo zero (positivo o negativo a seconda del segno). Esponente di tutti 1 con frazione di 0 rappresenta l'infinito positivo o negativo, utile per l'overflow. Esponente tutti 1 con frazione non nulla ha il valore Nan (Not a Number) e viene utilizzato per segnalare errori.

Aritmetica in virgola mobile:

Per somma e sottrazione è necessario allineare gli operandi aggiustando gli esponenti. Un operazione in virgola mobile può generare le seguenti eccezioni:

- ***overflow dell'esponente:*** esponente positivo eccede il massimo valore possibile
- ***underflow dell'esponente:*** esponente negativo minore del minimo valore possibile
- ***underflow del significando:*** alienando i significandi le cifre possono scorrere al di fuori del lato destro, si ricorre ad arrotondamenti
- ***overflow del significando:*** la somma di due significandi può provocare il riporto dei bit più significativo.

Somma e sottrazione:

- ***controllo dello zero:*** se uno dei due è zero
- ***allineamento delle mantisse****:* rendere uguali gli esponenti
- ***somma o sottrazione delle mantisse***
- ***normalizzazione del risultato:*** traslare a sinistra finché la cifra più significativo è diversa da 0

Moltiplicazione e divisione:

- *controllo dello zero*
- *somma degli esponenti*
- *sottrazione polarizzazione*
- *moltiplicazione/divisone operandi*
- *normalizzazione*

- *arrotondamento*

Considerazioni sulla precisone:

In genere i registri della Alu hanno più bit di quelli necessari per la mantissa, i bit è più a destra sono settati a 0 e permettono di non perdere bit se i numeri vengono shiftati a destra. Il risultato di ogni operazione sui significandi è generalmente memorizzato in un registro più lungo e quando viene riportato nel formato in virgola mobile bisogna arrotondare. Esistono quattro tecniche per questo:

- ***arrotondamento al più vicino:*** viene arrotondato al numero più vicino tra quelli rappresentabili
- ***arrotondamento a +infinto:*** arrotondato per eccesso
- ***arrotondamento a –infinito:*** per difetto
- ***arrotondamento a 0:*** i bit in eccesso vengono troncati

Linguaggio macchina:

Gli elementi che costituiscono un istruzione definiscono i passi richiesti per la sua esecuzione:

- ***Codice operativo:*** specifica l'operazione da eseguire (ADD, SUB)
- ***Riferimento all'operando sorgente:*** l'operazione può coinvolgere uno o più operandi sorgente
- ***Riferimento all'operando risultato:*** l'operazione può produrre un risultato
- ***Rifermento alla successivo risultato:*** indica alla cpu dove si trova la prossima istruzione da eseguire, in assenza di salti questo paramento è superfluo in quanto le istruzioni vengono eseguite sequenzialmente.

Data la complessità della programmazione binaria le istruzioni posso essere impartite usando dei codici mnemonici, le istruzioni binarie corrispondono a traduzioni in sigle e codici comprensibili all'uomo. Le istruzioni possono essere di elaborazione (aritmetiche/logiche), di memorizzazione, trasferimento (i/o) e controllo. Quindi quanti indirizzi sono necessari in un istruzione? Zero indirizzi quando tutti sono impliciti, viene utilizzata una pila (stack) di istruzioni con logica FIFO. Un solo indirizzo basta quando il secondo è implicito (l'accumulatore AC) ed è destinato a contenere uno degli operandi e per memorizzare il risultato. Due indirizzi nelle operazioni binarie in cui un indirizzo svolge da operando e risultato. Con tre indirizzi vengono specificati due operandi e un risultato.
Programmi formati da istruzioni con pochi indirizzi eseguono istruzioni più elementari e richiedono una Cpu meno complessa ma un maggior numero di istruzioni e tempi di esecuzione più elevati. Più indirizzi provocano istruzioni più complesse, vedi RISC vs CISC.

Tipi di operandi:

In alcuni casi occorre eseguire calcoli sui differimenti agli operandi per determinare l'indirizzo di memoria.

- ***Numeri:*** vi sono tipi interi, decimali, in virgola moglie. I decimali vengono rappresentati in modalità "decimale impaccata" ossia 4 bit per cifra (246 = 0010 0100 0110). Si tratta di una decodifica lenta ma evita il tempo della conversione.
- ***Caratteri:*** i caratteri ASCII usano 7 bit (128 caratteri) più un bit di parità.
- ***Dati logici:*** n bit invece di un singolo dato,

Per progettare un insieme di istruzioni è necessario considerare un repertorio (quante e quali), i tipi di dato utilizzabili, il formato (n° indirizzi), i registri e il metodo di indirizzamento.

Tipi di operazioni:

- ***Trasferimento dati:*** specifica la locazione degli operandi sorgente e destinazione (memoria, registri, in pila), la lunghezza dei dati da trasferire e il modo di indirizzamento. Le il trasferimento avvien tra registri l'operazione riguarda la sola Cpu, se riguarda anche la memoria allora va calcolato l'indirizzo di memoria (se memoria virtuale va tradotto in fisico), si determina se l'elemento indirizzato si trova in cache e in caso negativo si invia un comando alla memoria.
- ***Aritmetica:*** somma, sottrazione, moltiplicazione, divisione, incremento, decremento, negazione
- ***Logica:*** operazioni su bit (bit twidilling) NOT, AND XOR, shift logico, shift aritmetico, rotazione
- ***Conversione:*** cambiano il formato o operano sul formato dei dati
- ***Input/Output:*** i/o da programma, memory mapped e DMA
- ***Sistema:*** l'uso di queste istruzioni è riservato al sistema operativo e vengono eseguite solamente quando la macchina si trova in uno stato privilegiato
- ***Trasferimento di controllo:*** alcune istruzioni possono variare la sequenza di istruzioni da eseguire:
    - ***Salto condizionato:*** il salto viene intrapreso solo se si verifica una condizione
    - ***Salto incondizionato (skip):*** in genere utilizzata per scavalcare un istruzione, i bit per l'indirizzo di destinazione vengono sfruttati per altri scopi come l'incremento/decremento.
    - ***Chiamata di procedura:*** una procedura è una patte di codice interna ad un programma che viene invocata quando vi è la necessità di ripetere più volte la stessa parte di codice. Essendo modulari permettono di facilitare la programmazione. Prevedono una chiamata e un ritorno (salti). Le procedure possono essere chiamate da punti diversi del programma e consentono l'annidamento. L'indirizzo di ritondo viene salvato in un registro, all'inizio della procedura chiamata o in cima alla pila.

Indirizzamento:

Le architetture dei calcolatori offrono vari modi di indirizzamento quindi per determinare quale usare si usano due metodi, o lo si specifica nel codice operativo oppure si impiegano dei bit dell'istruzione.

- ***Immediato:*** l'operando è specificato nell'istruzione. Pro: nessun accesso in memoria per fetchare l'operando. Contro: valore limitato dalla dimensione del campo indirizzo.
- *Diretto:* il campo indirizzo contiene l'indirizzo dell'operando (ADD A), richiede un accesso in memoria per recuperare A ed è limitato uno spazio di indirizzamento limitato.
- ***Indiretto:*** il campo indirizzo contiene l'indirizzo di una cella M che contiene l'indirizzo dell'operando. Pro: con parole di lunghezza N si hanno 2^N entità diverse. Contro: due accessi in memoria per ottenere l'operando.
- ***Registro:*** l'operando è in un registro indicato nel campo indirizzo, lo spazio di indirizzamento è così limitato ma si hanno istruzioni più corte e fetch più rapido
- ***Registro indiretto:*** l'operando è in una cella di memoria puntata dal registro, grande spazio di indirizzamento (2^n) un accesso in memoria in meno rispetto all'indirizzamento indiretto.
- ***indirizzamento a pila:*** associato alla pila vi è un puntatore (SP stack pointer) il cui valore è l'indirizzo della cima della pila

- ***Spiazzamento:*** l'istruzione deve avere due campi indirizzo di cui uno viene usato direttamente mentre l'altro si riferisce a un registro il cui contenuto viene sommato al primo per produrre l'indirizzo effettivo.
    - ***Relativo:*** l'indirizzo dell'istruzione corrente viene sommato al campo indirizzo.
    - ***Registro-base:*** il registro referenziato contiene un indirizzo di memoria e il campo indirizzo contiene uno spiazzamento rispetto a tale indirizzo
    - ***Indicizzazione:*** il campo indirizzo rappresenta un indirizzo di memoria centrale e il registro referenziato contiene uno spiazzamento positivo da tale indirizzo

Formati delle istruzioni:

Il formato dell'istruzione definisce la disposizione dei suoi bit.

- ***lunghezza delle istruzioni:*** è condizionata dalla dimensione e organizzazione della memoria, dal bus, dalla Cpu e dalla sua velocità. Più codici operatici e operandi semplificano la programmazione , più modi di indirizzamento forniscono maggiore flessibilità quindi va trovato un compromesso.
- ***Allocazione dei bit:*** codici operativi a lunghezza variabile sono un compromesso tra dimensione dei campi codice operativo e indirizzi. Il modo di indirizzamento può essere indicato implicitamente; un minor numero di indirizzi può portare a programmi più lunghi.

Struttura della Cpu:

Le cpu integrano dei registri che costituiscono il livello più alto della gerarchia di memoria, essi si suddividono in:

- ***registri utente:*** memorizzano i dati da elaborare
    - ***generici:*** possono essere ad uso generale o dedicato, contengono dati (AC), indirizzi (spiazzamento). L'uso di registri specializzati riferisce implicitamente a quale tipo di registro si riferisca un certo operando necessitando un numero inferiore di bit limitando la flessibilità nella programmazione. Più registri si devono indirizzare e maggiore è il numero di bit necessari per identificare l'operando. Un numero compreso tra 8 e 32 sembra essere ottimale ma le architetture RISC ne usano qualche centinaio.
    - ***Codici di condizione:*** sono dei bit di flag che la cpu imposta a seguito di un operazione per memorizzare valori positivi/negativi, nulli, overflow. Sono generalmente non sono modificabili dal programmatore.
- ***di controllo e di stato:*** si tratta di registri non visibili all'utente e controllano le operazioni della cpu (PC, IR, MAR, MBR). LA PSW (program status word) include vari registri e mantiene le informazioni di stato come zero, riporto, overflow, abilitazione interrupt, modo supervisore

Ciclo esecutivo delle istruzioni:

Il ciclo indiretto è il sottociclo di un istruzione in cui si esegue il prelievo di indirizzi indiretti dalla memoria.
Le sequenza di eventi dipende dall'architettura della cpu ma in generale avvengono questi passaggi:

- ***instruction fetch:*** legge un istruzione dalla memoria, il PC, che contiene l'indirizzo della successiva istruzione da prelevare, viene scritto nel MAR e impostato sul bus indirizzi, viene letta la memoria e il risultato scritto sul bus dati, copiato nell'MBR e trasferito nell'IR, il PC viene incrementato per il prossimo prelievo

- ***data fetch:*** si esamina l'IR, se contiene opcode che richiede l'indirizzamento indiretto si esegue un ciclo di indirettezza, gli N bit più a destra dell'MBR (il riferimento all'indirizzo) vengono trasferiti nel MAR, l'unità di controllo legge la memoria e trasferisce il risultato (indirizzo dell'operando) nell'MBR.
- ***Execute:*** può assumere varie forme, può includere lettura/scrittura della memoria, i/o, trasferimenti tra registri e operazioni Alu
- ***Interrupt:*** il PC viene trasferito nell'MBR per essere scritto in memoria, la locazione di memoria viene caricata nel MAR e il contenuto dell'MBR viene scritto in memoria. Infine nel PC viene scritto l'indirizzo della prima istruzione della routine che gestisce l'interrupt. Queste operazioni sono necessarie per permettere la ripesa del lavoro interrotto dall'interrupt.

Pipeline:

La tecnica del pipelining introduce il parallelismo tra le fasi del ciclo esecutivo. Immaginando di avere le sole fasi di prelievo ed esecuzione, durante la seconda fase è possibile eseguire prelevare e bufferizzare l'istruzione successiva. In questo modo si esegue il prefetch delle istruzioni o fetch overlap. Poiché la fase di escuzione è più lunga del prelievo, le prestazioni non raddoppiano inoltre in caso di salto condizionato l'istruzione successiva è indeterminata fino alla fine dell'execute; a questo scopo esistono varie politiche di prefetch (in genere si preleva l'istruzione successiva al salto). Per aumentare il livello di parallelismo si hanno pipeline a più stadi:

- ***Fetch (FI):*** legge l'istruzione da eseguire e la pone in un buffer
- ***Decodifica (DI/ID):*** determina il codice operato e gli identificatori degli operandi
- ***Calcolo indirizzi operandi (CO):*** calcola l'indirizzo effettivo degli operandi
- ***Fetch operandi (FO):*** legge gli operandi in memoria
- ***Esecuzione (EI):*** effettua l'operazione
- ***Scrittura (WO):*** memorizza il risultato

Nella realtà alcuni stadi come FI, FO, WO non possono avvenire simultaneamente in quanto necessitano di un accesso memoria (conflitti di memoria). Ogni stadio causa dell'overhead per il trasferimento dei dati tra i buffer e le quantità di circuiti di controllo richiesta per gestire le dipendenze della memoria e dei registri aumenta con l'aumentare del numero degli stadi.
Per calcolare il tempo t necessario per far avanzare di uno stadio le istruzioni attraverso la pipeline si usa la formula: $t=\max_i[t_i]+d=t_m+d$ con $1 <= i <= k$ dove t_m è il ritardo dello stato più oneroso, k il numero di stadi della pipeline, d il ritardo di commutazione di un registro richiesto per l'avanzamento di segnali e dati da uno stadio al successivo.
Tk il tempo totale per eseguire n istruzioni richiesto da una pipeline con k stadi:
$Tk = [k + (n-1)]t$
Il fattore di velocizzazione (speedup) è $Sk = T1/Tk = nk / [k + (n-1)]$

Possono verificarsi delle condizioni che impediscono il naturale parallelismo della pipeline, esse sono dette "stalli" e possono essere causati da:

- ***sbilanciamento delle fasi:*** diversa durata delle fasi e istruzioni diverse; le soluzioni possono essere decomporre in sottofasi le operazioni più onerose e duplicare gli esecutori delle fasi più onerose per operare in parallelo
- ***problemi strutturali:*** sovrapposizione delle fasi di istruzione che causano conflitti di accesso alle risorse (come conflitti di memoria). Si possono suddividere le memorie per esempio introducendo una cache istruzioni e una dati si ha un accesso parallelo, inoltre si può introdurre la fase non operativa (nop).

- ***dipendenza dei dati:*** l'operazione successiva dipende dai risultati della precedente. Un dato modificato nella fase EI può dover essere utilizzato dalla fase FO dell'istruzione successiva. Esempi: ReadAfterWrite, WriteAfterWrite, WriteAfterRead. Soluzioni: nop, data forwarding (prelievo del dato direttamente dall'uscita dell'Alu), risoluzione a livello di compilazione, riordino delle istruzioni (pipeline scheduling)
- ***dipendenza dai controlli:*** istruzioni come i salti condizionati che invalidano la sequenzialità delle istruzioni. Sono circa il 30% del totale medio di un programma. E' possibile mettere in stallo la pipeline fino al calcolo dell'istruzione successiva ma questo approccio è poco efficiente; oppure mediante apposita logica di controllo (hardware aggiuntivo) si possono individuare le istruzioni critiche per anticiparne l'escuzione. Alcune soluzioni per i salti condizionati:
    - ***flussi multipli (multiple streams):*** replicando le parti iniziali della pipeline è possibile caricare l'istruzione successiva a quella di salto e l'istruzione destinazione del salto, potrebbero però crearsi dei conflitti di accesso alle risorse e la presenza di salti condizionati in sequenza richiederebbe altre due pipeline per ogni salto annidato.
    - ***Prelievo anticipato della destinazione (prefetch branch target):*** si effettua il fetch anticipato della istruzione di destinazione del salto ma non evita l'eventuale svuotamento della pipeline
    - ***Buffer circolare (loop buffer):*** una memoria piccola e molto veloce mantiene le ultime n istruzioni prelevate, in caso di salto si controlla se l'istruzione destinazione è già presente in buffer; in caso di salto di poche istruzioni esse saranno già presenti nel buffer e un ciclo relativamente piccolo potrebbe eseguire le sole istruzioni contenute nel buffer evitando ulteriori fetch dalla memoria.
    - ***Predizione dei salti (branch prediction):*** si cerca di prevedere se il salto sarà intrapreso, gli approcci statici prevedono: si salta sempre, mai, in base al codice operativo; i metodi dinamici migliorano la predizione in base al programma eseguito utilizzando un bit taken/not taken (memorizzati in una locazione temporanea associano uno o più bit che codificano la storia recente) o una tabella della storia dei salti (piccola memoria associata allo stadio fetch, contiene tre elementi: indirizzo istruzione salto, destinazione salto, bit di storia che descrivono lo stato dell'uso dell'istruzione).
    - ***Salto ritardato (delayed branch):*** utilizza gli stadi inattivi a causa di stallo per svolgere lavoro utile; la cpu esegue sempre l'istruzione che segue il salto e solo dopo altera se necessario la sequenza di istruzioni, l'istruzione che segue quella di salto si dice essere posta nel branch delay slot dove il compilatore cerca di allocare la più opportuna

RISC:

L'architettura RISC si caratterizza per avere un gran numero di registri generici, un insieme di istruzioni limitato e una pipeline particolarmente ottimizzata. RISC rappresenta un inversione di tendenza infatti in passato si cercava di sviluppare linguaggi sempre più complessi e potenti ma veniva a crearsi un enorme gap semantico tra comandi ad alto livello e istruzioni macchina. A seguito di svariati studi si è capito che i set di istruzioni andavano semplificati e bisognava lavorare sull'ottimizzazione delle cpu.
L'uso di un ampio banco di registri permette di salvare molte variabili scalari locali in locazioni veloci ed interne alla cpu (no uso bus) impiegando anche indirizzi più corti. Il compilatore avrà il compito di massimizzare l'uso dei registri allocando le variabili più utilizzate. Ogni chiamata a procedure cambia lo scope delle variabili quindi bisognerà lavare

in memoria le variabili locali, passare i parametri al ritorno le variabili del programma chiamate vanno ricaricate nei registri e i risultati vanno passati al programma chiamante. Tramite dei piccoli gruppi di registri ciascuno assegnato a una differente procedura è possibile organizzare il banco registri in modo da ottimizzarne l'utilizzo, infatti si è visto che al maggior parte delle procedure coinvolgono un numero ristretto di parametri e non presentano un grado di annidamento elevato. Ogni gruppo è suddiviso in: registri dei parametri, registri locali memorizzano le variabili locali, registri temporanei si sovrappongono con quelli del gruppo successivo (fisicamente sono gli stessi) in questo modo è possibile il passaggio di parametri senza trasferimento fisico.

Buffer circolare:

- quando avviene una chiamata il puntatore alla finestra corrente (Current Window Pointer) viene aggiornato per mostrare la finestra di registri corrente attiva
- se si esaurisce la capacità dl buffer viene generato un interrupt e la finestra più vecchia viene salvata in memoria
- un puntatore (Saved Windows Pointer) indica dove si deve ripristinare l'ultima finestra salvata in memoria principale

Per le variabili globali o vengono allocate in memoria dal compilatore (inefficiente se riferite di frequente) o dei gruppi di registri vengono dedicati alle variabili locali.

Compilatori ottimizzanti:

Per mantenere gli operandi nei registri per la maggior parte del tempo e minimizzare il numero di accessi alla memoria il compilatore esegue degli algoritmi. Ogni variabile viene associata ad un registro simbolico virtuale, quindi i registri simbolici vengono mappati sui reali; i registri simbolici il cui uso non si sovrappone temporalmente possono condividere lo stesso registro reale, le variabili per cui non è possibile utilizzare un registro reale vengono mantenute in memoria. Questa tecnica è derivata da quella della colorazione di un grafo, si vuole assegnare dei colori ai suoi nodi in modo che nodi adiacenti non abbiano lo stesso colore e il numero di colori usati sia minimo. Se due registri sono attivi nello stesso momento allora vengono congiunti da un segmento, si cerca di colorare il grafo con n colori dove n è il numero di registri reali, i nodi che condividono lo stesso colore possono essere assegnati allo stesso registro, i nodi che non possono essere colorati vengono posti in memoria.

CISC vs RISC:

Nelle architetture CISC le istruzioni macchia sono difficili da sfruttare, l'ottimizzazione è più difficile, i programmi sono più corti ma non è detto che occupino meno memoria la quale sta diventando sempre più economica, i codici operativi sono più lunghi, le unità di controllo più complesse. Di contro il RISC esegue un istruzione per ciclo, le istruzioni vengono implementate in hardware; effettua operazioni da registro a registro, quindi meno accessi in memoria; l'indirizzamento è più semplice, indirizzamento registro; formato istruzioni è semplificato, si utilizzano uno o due formati a lunghezza fissa allineata alla grandezza della parola, le posizioni dei campi sono fisse e la decodifica dei codici operativi e l'accesso agli operandi nei registri possono avvenire simultaneamente.
Di fatto non esiste un vincitore perché non è mai stato effettuato un vero confronto che escludesse variabili non controllabili. Non esiste un set completo di programmi di test e molti test sono stati svolti su prototipi e versioni semplificate. Ecco perché la maggior parte delle Cpu commercializzate adottano caratteristiche da entrambi le filosofie.

MIPS:

La MIPS è un esempio di architettura RISC con pipeline ottimizzata. Le istruzioni occupano 32 bit e non si fa uso di codici di condizione sostituiti da flag memorizzati in un registro generico semplificando la circuiteria. Le istruzioni a formato fisso facilitano il fetch e la decodifica. I registri sono 32 da 32 bit (r0 contiene sempre 0) e le operazioni avvengono sempre tra registri. Load e Store sono istruzioni per trasferire dati tra memoria e registri, tutte le altre istruzioni operano esclusivamente sui registri. Nei registri è possibile scrivere byte, mezze parole o parole estendendo con 0 gli eventuali bit non coinvolti. E' ammesso l'indirizzamento immediato(diretto), con spiazzamento, indiretto registro (spiazzamento a 0) e assoluto (registro 0 come base).

Pipeline MIPS:

Le fasi sono 5:

- ***Instruction fetch (IF):*** legge l'istruzione da eseguire ed incrementa il PC
- ***Instruction decode/register fetch (ID):*** in parallelo decodifica l'istruzione e legge i registri
- ***Execution (EX):*** la Alu esegue le operazioni sugli operandi che possono essere di tre tipi:
    - ***Riferimento alla memoria:*** la Alu somme il registro base e l'offset per formare l'indirizzo effettivo
    - ***Registro-registro:*** esegue i calcoli sui valori letti dai registri
    - ***Registro-immediato:*** esegue le operazioni specificate dal codice operativo sul primo valore letto dal registro
- ***Memory access (MEM):*** nella Load la memoria effettua una lettura usando l'indirizzo calcolato nel ciclo EX, nella Store la memoria scrive i dati del secondo registro
- ***Write-back (WB):*** istruzione registro-registro o Load: scrive il risultato nel registro, sia che provenga dalla memoria (Load) o dalla Alu (istruzione Alu)

MIPS esegue una fase in un ciclo di clock; i pipeline registers si occupano di bufferizzare i dati tra una fase e l'altra memorizzando dati e segnali di controllo. Nella fase ID è possibile individuare tutte le dipendenze dai dati e determinare il data forwarding da effettuare.
Il forwarding è l'operazione che sfrutta una scorciatoia per trasferire un dato da dove viene prodotto a dove viene richiesto, questo riduce il numero di stalli
E' possibile effettuare il forwarding da:

- *output ALU*
- *memoria dati*

Verso:

- *input ALU*
- *input memoria dati*
- *comparatore con 0*