Con l'espressione **architettura di von Neumann** (o macchina di von Neumann) ci si riferisce a uno schema di progettazione di calcolatori elettronici che prende nome dal matematico John von Neuman.
Il modello di Von Neumann è quanto di più prossimo alla reale struttura di un calcolatore; può essere considerato (in via esemplificativa) il modello base di tutti i moderni calcolatori.
Lo schema si basa su cinque componenti fondamentali:

1. CPU o unità di lavoro che si divide a sua volta in
    1. Unità operativa, nella quale uno dei sottosistemi più rilevanti è l'ALU (Arithmetic Logic Unit)
    2. Unità di controllo
2. Unità di memoria, intesa come memoria di lavoro o memoria principale (RAM, Random Access Memory)
3. Unità di input, tramite la quale i dati vengono inseriti nel calcolatore per essere elaborati
4. Unità di output, necessaria affinché i dati elaborati possano essere restituiti all'operatore
5. Bus, un canale che collega tutti i componenti fra loro

Il funzionamento in sequenza della macchina di Von Neuman segue i seguenti passi :
Fase di fetch:
• preleva dall'esterno una stringa di bit che indica il prossimo passo da fare (macchinaprogrammabile);
• Esegue, quindi, uno dopo l'altro una sequenza di passi (programma).
Fase di esecuzione:
• svolge quanto richiesto;
• accede all'esterno per scambiare dati.
Fase di decodifica:
• interpreta la stringa di bit come istruzione macchina.

Per essere programmabile, la CPU deve scoprire i propri "compiti" volta per volta in modo tale da poter variare il proprio risultato sulle esigenze esterne.I "compiti":
• sono sequenze di stringhe di bit:
– ogni stringa un passo o una istruzione macchina che la nostra CPU deve svolgere;
– ogni sequenza un compito o programma.
• elaborano informazioni (dati) codificate mediante stringhe di bit.
Il contenitore che ci consente di scambiare le informazioni con la CPU di tante stringhe di bit che possa rispettandone i tempi di lavoro molto brevi e la c.d MEMORIA DI LAVORO. Array di "celle" contenenti ciascuna una parola. La singola cella è individuata dal proprio indirizzo.
La CPU segnala alla memoria:
• la cella a cui è interessata (mediante l'indirizzo);
• il tipo di operazione che intende svolgere:
– prelievo o Lettura del contenuto della cella;
– modifica o Scrittura del contenuto nella cella.
CPU e memoria si scambiano il contenuto della cella, secondo la direzione richiesta dalla CPU.
La CPU svolge il ruolo Master: cioè decide quando e cosa fare e che tipo di trasferimento
La memoria subisce avendo un ruolo Slave cioè rispondere alle richieste della CPU.
La CPU interagisce con il resto del mondo elettronico (memoria e interfacce) mediante una serie di linee parallele (BUS).
• Alcune linee:
– sono dedicate al trasferimento di bit.
– sono dedicate all'indicazione della cella cui la CPU fa riferimento.
– servono a orchestrare le interazioni fra CPU e resto del mondo elettronico.
• I numeri di linee (dimensioni del bus) sono correlati alle prestazioni della CPU.

I modi di indirizzamento sono le diverse modalità disponibile nell' linguaggio macchina per recuperare i dati necessari per l'esecuzione delle istruzioni macchina.

La CPU LC-2 è una macchina RISC: il suo ISA (Instruction Set Architecture) è infatti costituito da istruzioni macchina che occupano ciascuna una sola cella di memoria (istruzioni a 16 bit, da cui la dimensione del registro IR: Instruction Register) e con codice operativo (opcode) di soli 4 bit. Ciò consente dunque di codificare solo 2_4=16 diverse istruzioni macchina.
L'approccio RISC è evidente anche nell'adozione di un numero limitato di modi di indirizzamento:

- immediato l'operando è contenuto nell'istruzione macchina;
- diretto l'istruzione macchina fornisce l'indirizzo della cella di memoria contenente l'operando;
- indiretto l'istruzione macchina fornisce l'indirizzo di una cella di memoria contenente a sua volta l'indirizzo della cella di memoria contenente l'operando;
- base+offset l'istruzione macchina indica un registro GPR e un valore numerico da sommare al contenuto del registro GPR per ottenere l'indirizzo della cella di memoria contenente l'operando.
-

In base ai numeri di accessi alla memoria effettuati per recuperare i dati viene scelto un modo pressoché un altro da utilizzare:

immediato:durante la fase di fetch si preleva anche le informazioni necessarie per eseguire l'istruzione

diretto:richiede al unita centrale un nuovo accesso a memoria per prelevare il dato da utilizzare

indiretto:sono necessari due ulteriori accessi a memoria ;il primo per prelevare l'indirizzo del operando ,il secondo per prelevare l'operando vero e proprio

base+offset:simile all'accesso diretto,serve un accesso a memoria per prelevare il dato ma l'indirizzo viene costruito sulla base dell' contenuto di un registro della CPU che quindi può essere dinamicamente modificato a cui sommare un offset.

Interrupt vettorizzato

La CPU è dotata delle linee INTREQ e INTACK.
Alla ricezione di un interrupt:
• salva il valore del PC;
• disabilita il riconoscimento di ulteriori interrupt;
• attiva INTACK;
• attende sul Data Bus la comparsa di un identificativo a 8 bit - inserito dall'interfaccia a periferica - che usa (come per l'istruzione TRAP) come indice in un vettore di interrupt:
– tabella di celle di memoria, una associata a ogni possibile sorgente di interrupt;
– ogni cella contiene l'indirizzo di inizio della routine di risposta all'interrupt associato.
☐ Il tempo di riconoscimento della sorgente di interrupt è minimizzato:
– non serve polling.
☐ Le interfacce a periferica si complicano:
– ogni interfaccia deve essere capace di generare il proprio identificativo sul Data Bus;
– ogni interfaccia deve "sapere" il proprio identificativo.
☐ Non abbiamo risolto i problemi di priorità:
– se la CPU viene riabilitata a sentire gli interrupt, può essere interrotta da chiunque.

Un Controllore di Interruzioni Programmabile (PIC, Programmable Interrupt Controller) è un dispositivo hardware che consente di gestire interruzioni vettorizzate con priorità per conto di un processore. Con il PIC l'interfaccia comunica al PIC stesso la richiesta di interrompere l'unita centrale. Se l'interfaccia e abilitata a farlo ,il PIC attiva la linea di richiesta a INTREQ. Quando riceve INTACK il PIC comunica l'identificativo sull' Data Bus : gli identificativi delle periferiche sono gestiti dal PIC, senza aggiunta di complessità per le interfacce. Inoltre se il PIC riceve più richieste di interrupt da diverse periferiche, dà precedenza a quella più prioritaria.

Funzionamento del PIC

Se la periferica al quale l'interfaccia e collegata segnala che sia verificato un evento l'interfaccia mediante un segnale elettrico segnala al PIC di essere pronta a essere servita ,di richiedere il servizio da parte della CPU. Nel caso in cui l'interfaccia e abilitata a interrompere la CPU il PIC genera la richiesta di INTREQ e quando la CPU risponde con l'accettazione di INTACK il PIC si preoccupa di generare sull' bus dati il vettore di IntVect(identificativo a 8 bit di quale periferica e quella interrompente)onde a consentire all' Unita centrale di saltare direttamente alla rutine di risposta alla periferica opportuna. Questo giro di segnali che attivano un interrupt vettorizzato in presenza dell' dispositivo programmabile di controllo delle interruzioni.
I registri del PIC sono normali registri di interfaccia accessibili alla CPU in modo di consentire alla CPU di programmare il PIC ovvero al programma di gestione del interazione ingresso/uscita di definire come il PIC deve comportarsi.
Per fare questo essenzialmente servono 3 registri:

IVR(Interrupt Vector Register): contiene l'identificativo associato a ciascuna periferica collegata
IPR Interrupt Priority Register: contiene le informazioni necessarie per stabilire l'ordine di priorità delle periferiche;
IMR Interrupt Mask Register: contiene le informazioni per sapere quali periferiche possono generare interruzioni e quali no.

Una possibile struttura generica di una unita di controllo cablata può essere la seguente: per quanto riguarda capire di quale istruzione si tratta possiamo prendere i due bit più significativi dell' IR ed inserirli come ingressi in un decoder 2 a 4 che attivi 1 delle 4 linee(L/S/A/B) a seconda che l'istruzione presente nell'IR sia (load/store/add/branch)in questo modo la decodifica dell'istruzione viene fatta da un decoder hardware. Per quanto rigurada il CC e sufficiente far uscire il valore presente nell' bistabile Z e renderlo disponibile alla CU infine per quanto riguarda il passo di avanzamento possiamo immaginare di realizzare un contatore che vienne incrementata dall', Clock della CU e ancora una volta un decoder che trasforma il valore dell' conteggio nell' accensione successiva di una delle 10 linee (S_0-S_9) che ci dicono a quale step siamo arrivati nell' avanzamento della nostra sequenza di comandi.
Caratteristiche della CU cablata:

- E una struttura particolarmente molto efficiente perché la rete combinatoria Logic array e il modo più rapido per generare le uscite a partire dagli ingressi .
- Quindi assicura la massima velocità di esecuzione
- E particolarmente adatta a CPU RISC:con poche istruzioni perche la complessità del Logic array della rete combinatoria e dominabile e questo e il motivo per cui inizialmente le macchine RISC hanno avuto un grosso successo a livello hardware ,riuscivano ad avere una notevole velocità proprio grazie alla realizzazione della CU cablata.

Ha pero delle controindicazioni la CU cablata:

- Non e di facile modifica , evoluzioni del architettura che portino a modifica della struttura CPU richiedono la riprogettazione completa della CU cablata.
- Non e particolarmente facile usare una CU cablata per macchine CISC (macchine con set di istruzioni complesso)perche il Logic array che realizza l'espressioni combinatorie tende a esplodere esponenzialmente col crescere dei ingressi e delle uscite
- Solo recentemente ,grazie all evoluzione tecnologica,dell' integrazione sull' silicio e diventata applicabile a CU complesse ,CU di unita centrali (CISC).Negli anni precedenti la cosa era poco proponibile per la eccessiva complessità hardware di una CU cablata in presenza di un set di istruzioni complesso.

La struttura della CU microprogrammata si basa su una memoria di microprogramma solitamente di una memoria a sola lettura ROM nella quale ogni parola contiene quello che viene definita una microistruzione. La microistruzione e composta da una parte di microcomandi che sono di fatto gli output CU(comandi a registri, ALU, Control Bus) e l'alta parte della microistruzione e ciò che serve per decidere quale prossima microistruzione al interno del microprogramma dovremo prelevare ed eseguire .Traducendo il comportamento della CU in un comportamento sequenziale di passi o microistruzioni in cui e possibile mediante i bit dell' CC e mediante i bit che ci dicono il codice operativo acquisito durante la fase di fetch come muoverci al interno di questo microprogramma (micro perche e contenuto all' interno dell' CU ;programma perche ha la tipica struttura di esecuzione sequenziale di passi) .Esiste anche un microprogram counter che e un registro interno alla CU che serve a scandire in sequenza le microistruzioni salvo momenti nei quali abbiamo dei microsalti delle modifiche del percorso di analisi di scansioni delle microistruzioni all' interno della memoria del microprogramma .
Le principali caratteristiche della CU microprogrammata :

- E una struttura particolarmente regolare ,la ROM cambia le sue dimensioni al variare della complessità del microprogramma ma rimane essenzialmente una struttura molto regolare dal punto di vista circuitale e cosi si può dire anche dei supporti esterni come microprogram counter e la logica di gestione dei microsalti
- E particolarmente adatta a CU complesse come quelle necessarie per CPU(CISC) quando la tecnologia hardware non e cosi evoluta da consentire il progetto di CU cablate estremamente sofisticate e comunque facilitando il progetto della CU perche le tecniche di progettazione dell' software si possono di fatto riapplicare anche al progetto dell' microcodice della CU .
- Per questo l'approccio microprogrammato era particolarmente usato negli anni '80 per consentire alla realizzazione hardware di CPU complesse quindi di CU che richiedevano di gestire molti segnali di I/0.
- Ha problemi di velocità di esecuzione dovuti alla presenza della memoria di microprogramma e sempre necessario il ciclo di lettura del prossimo passo della prossima microistruzione prima di poter emettere i comandi che la CU deve emettere a partire dagli ingressi alla CU stessa e questo fa si che li step della CU microprogrammata siano decisamente più lenti degli step della CU cablata .
- L'approccio cablato tende a soppiantare l'approccio microprogrammato non appena la tecnologia di integrazione lo consenta

Una pipeline si può efficacemente immaginare come una catena di montaggio, attraverso la quale le istruzioni vengono processate per essere eseguite. Ogni istruzione, cioè, viene scomposta in operazioni più semplici, eseguibili dai vari stadi della pipeline, e dunque percorre tutta la "catena di montaggio" fino alla sua completa esecuzione. Il

vantaggio di tutto ciò è che una volta che un'istruzione abbandona uno stadio della pipeline, tale stadio può occuparsi dell'istruzione successiva, la quale dunque verrà completata dopo la precedente in un tempo sensibilmente inferiore a quello della sua completa esecuzione. Una struttura a catena di montaggio (pipeline) e un ottimo approccio per velocizzare l'esecuzione non tanto della singola istruzione quanto del numero di istruzioni in corso di lavorazione e quindi che verranno completate per unita di tempo con un incremento netto dell' [MIPS] che la nostra macchina e in grado di portare a termine.

L'incremento delle prestazioni e dovuto tipicamente ed esclusivamente ad un intervento architetturale:la tecnologia di realizzazione dell'integrato circuito CPU e quindi la sua frequenza di lavoro non deve cambiare.

A parità di tecnologia un approccio architetturale pipeline consente guadagni di prestazione di un fattore da 2 a 8 .

I problemi nell'adozione di un modello a pipeline sono legati in generale alle cosi dette dipendenze (dependencies) cioè al fatto che durante l'esecuzione di programmi da parte di una CPU pipeline ci sono relazioni di dipendenza fra le diverse istruzioni dell' programma o fra le diverse risorse che il programma richiede di utilizzare che comportano conflitti nella catena di montaggio nella pipeline e quindi rallentamento della sua attività con perdita di efficacia .
Control dependency: il flusso di esecuzione delle istruzioni interrompe il normale comportamento sequenziale.
Data dependency:due istruzioni vicine utilizzano uno stesso dato.
Resource dependency: due istruzioni entrano in conflitto per una risorsa del sistema.
I legami fra le istruzioni in esecuzione e i loro conflitti di accesso alle risorse possono provocare rallentamenti della pipeline dovuti a:
• control dependencies;
• data dependencies;
• resource dependencies.
Si riesce a ridurre le inefficienze ricorrendo a soluzioni architetturali anche complesse:
• branch prediction;
• data forwarding;
• CPU con cache di primo livello separata per istruzioni e variabili(macchina di Harvard).

Il principio di località
Se all'istante t la CPU genera l'indirizzo di memoria xNNNN, è molto probabile che nell'immediato futuro generi di nuovo lo stesso indirizzo xNNNN o indirizzi vicini ("locali") all'indirizzo xNNNN.
Le motivazioni a cui e dovuta la località
Località spaziale:
– il fetch delle istruzioni procede normalmente per celle consecutive quindi se al istante t abbiamo fatto fetch dell' istruzione al indirizzo xNNNN e molto probabile che al prossimo passo si vada al istruzione situata nella cella immediatamente successiva
– i programmi soppratuto se sono di una certa dimensione sono organizzati in moduli,in procedure e le variabili del singolo modulo sono tipicamente memorizzate in spazi di memoria vicini.
Località temporale:
– l'essenza della programmazione sono i cicli: l'esistenza di gruppi di istruzioni che vengono scritte dall' programmatore o dal traduttore automatico una volta ed eseguite ripetutamente centinaia ,miglia di volte dal calcolatore ;questo significa che le istruzioni e le variabili usate nei cicli vengono "ripassate"ripetutamente a ogni iterazione dell' ciclo.
Dal momento che il principio di località e sicuramente una verità universale per il calcolatore va sfruttato. Lavorando su base statistica : quando la CPU genera un indirizzo di memoria,portiamo il contenuto della cella richiesta dell' CU e un certo numero di celle vicine (blocco) in una memoria(R/W):
decisamente più veloce della DRAM;
ovviamente più piccola,perche per essere più veloce deve essere più costosa da realizzare il singolo bit e richiederà una tecnologia sicuramente più costosa e quindi a parità dell' numero di bit il costo di questa memoria sarà decisamente più elevato .
Chiamiamo questa memoria cache:
per derivazione dalla parola francese cache (nascosto) perché l'esistenza di questa memoria di fatto non è nota né al programmatore,né alla CPU;
serve solo a velocizzare gli accessi a memoria.
Grazie alla località degli accessi a memoria da parte della CPU:
possiamo copiare in una memoria ad alte prestazioni (cache) le celle richieste, che hanno maggiore probabilità di essere usate di nuovo

Principio di funzionamento delle memorie cache

L'utilità della memoria cache (ripostiglio) nasce dal fatto che, sebbene il processore abbia velocità di elaborazione molto elevata (dell'ordine dei GHz, quella del clock), la memoria di sistema ed i bus di trasferimento lavorano con velocità inferiore.

Quando la veloce CPU è chiamata ad elaborare dati è, quindi, costretta ad aspettare che questi arrivino dai suoi bus e dalla sua memoria esterna di sistema; in questo modo le prestazioni complessive degradano inevitabilmente..

Per questo è stata inventata la cache memory, che trova posto tra il processore e la memoria RAM; si tratta di una memoria di piccole dimensioni ma particolarmente veloce; la sua velocità può variare infatti da quella di clock (se di primo livello) a valori comunque superiori a quella del bus.

In questo modo, almeno nell'immediato e con sufficiente probabilità, la CPU troverà in essa anche i dati necessari in seguito, senza dover attendere troppo.

Fin dalle prime architetture è stata prevista la presenza di cache tra CPU e memoria, direttamente sulla scheda madre e, data la grande efficienza di questo meccanismo, ben presto si è pensato di introdurre parte di essa addirittura dentro il processore; per velocizzare lo scambio di dati tra memoria e processore sono oggi disponibili:

• cache L1 (di primo livello) - qualche KB.
La frequenza del processore, però, è cresciuta ancora, e la sua differenza rispetto alla DRAM si è enfatizzata:
• cache L2 (di secondo livello) esterna al processore - qualche centinaio di KB.
E se le cose degenerano...:
• cache L3 (di terzo livello) esterna al processore - qualche decina di MB!
Grazie alla località degli accessi a memoria da parte della CPU:
• possiamo copiare in una memoria ad alte prestazioni (cache) le celle richieste, che hanno maggiore probabilità di essere usate di nuovo;
• possiamo creare una gerarchia di cache via via più grandi e più lente man mano che ci allontaniamo dalla CPU e ci avviciniamo alla memoria di lavoro;
• le celle con più alta probabilità di riutilizzo sono ricopiate nella cache a bordo della CPU;
• tutte le celle disponibili sono presenti in memoria di lavoro.

La politica Tag Associative definisce a priori una corrispondenza univoca fra :
• gruppo di blocchi in memoria di lavoro (MdL);
• e blocco di possibile destinazione di ciascuno dei blocchi di quel gruppo in cache (MC).
Per esempio l'indirizzo generato dalla CPU (16 bit) ha la seguente struttura:
I 4 bit meno significativi ci dicono a quale parola dell' blocco siamo interessati,i 7 bit immediatamente più significativi ci dicono a quale unita di gruppo l'unita centrale vuole fare riferimento e i 5 bit più significativi dei 16 ci dicono il numero di blocco nel gruppo quindi a quale dei 32 blocchi(5 bit → 25=32) che ci sono in ogni gruppo l'unita centrale intende accedere.
Caratteristiche Tag Associative
E una politica semplice :
il blocco richiesto dalla CPU può infatti trovarsi solo in un blocco di cache e quindi la scoperta se abbiamo fatto HIT/MIST e rapida e priva di problemi , basta andare a leggere il Tag associato al unico blocco di cache dove può trovarsi il blocco di MDL richiesta dall' unita centrale ,la risposta e una lettura dal tag che ci da si/no e quindi scoprire HIT/MISS e un unico accesso alla memoria di tag.
In caso di MISS il blocco richiesto può essere ricopiato in un'unica posizione quindi non ci sono particolari gradi di liberta o particolare scelte da fare dobbiamo sostituire il blocco che non ci interessa con il blocco interessato
Purtroppo e una politica non ottimizzata infatti ogni blocco di MC ottimizza solo localmente l'accessibilità ai blocchi di MDL cui e collegato ovvero ai blocchi di MDL associato a tale blocco di memoria cache .In ogni blocco di memoria cache c'e il blocco dell' gruppo associato a tale blocco di cache più recentemente richiesto da parte dell' unita centrale .Lo sfruttamento dei blocchi di MC non è uniforme .

Struttura e caratteristiche della memoria cache di tipo fully associative.

Per rimuovere l'ipotesi di allocazione fissa di blocchi di memoria di lavoro in cache che il problema della poca efficienza della politica tag associative la politica fully associative prevede un accoppiamento libero. Qualsiasi blocco di memoria di lavoro puo andar'a finire in qualsiasi blocco di cache ,questo significa che sparisce il concetto di gruppo e i blocchi di memoria di lavoro non sono piu raggruppati in gruppi aventi la proprietà di condividere uno stesso possibile blocco di destinazione in cache. Per esempio l'indirizzo generato dalla CPU (16 bit) ha la seguente struttura:Sparisce la distinzione tra bit di gruppo e bit di blocco ;tutti i 12 bit che non indicano una parola all interno dell' blocco cioè tutti tranne i 4 meno significativi ci dicono il numero di blocco in memoria di lavoro NBMdl quindi a quale dei 12 bit → 212=4096 blocchi di memoria di lavoro l'unita centrale e interessata .
Per poter verificare rapidamente se il blocco richiesto è in cache:
• memoria dei tag deve essere ad accesso associativo:
– presentare il valore cercato;

– ottenere in un tempo di accesso l'indirizzo della cella che
lo contiene (oppure segnalazione di assenza).
Per poter decidere dove scrivere il blocco cercato se non è presente:
• politica LRU (Least Recently Used);
• un contatore a saturazione per ogni blocco di MC:
– viene azzerato quando si accede al blocco associato;
– incrementato di 1 se si accede a un altro blocco;
– almeno un contatore sempre saturo (11...11).
Struttura memoria:
Le celle della memoria sono realizzate con una rette combinatoria basata su porte OR esclusive seguite da negatori e porte AND che propagano il confronto tra il contenuto del singolo bit e il contenuto dell' associative register , ogni bit dell' registro associativo viene distribuito in parallelo a tutta la colonna di celle e quindi ogni cella può iniziare a confrontare i propri bit con i bit del registro associativo e propagare l'esito di questi confronti .Le reti combinatorie compaiano a 0 se il contenuto della cella non corrisponde al registro associativo 1 se c'e invece corrispondenza .
Caratteristiche principali politica Fully Associative :
E senz'altro una politica ottimizzata i blocchi presenti in cache sono sempre quelli che nell' recente passato sono stati i più richiesti dalla CPU
E quindi otteniamo una ottimizzazione globale dello sfruttamento dei blocchi di cache omogeneamente utilizzati da tutti i blocchi di memoria di lavoro più recentemente richiesti e con maggiore probabilità di riutilizzo a breve.
Purtroppo e una politica complessa e costosa , la ricerca del blocco richiesto e la verifica se si trova in cache implica il ricorso a memoria associativa per i tag
Serve poi per implementare la politica di last recently used di eliminazione del blocco da più tempo non usato prevede l'uso dei contatori a saturazione che devono anch'essi essere accessibili in modo associativo .

Confronto fra politica store in e politica store thru nella realizzazione di memorie cache.

Se la CPU fa una scrittura in memoria: il dato in cache viene modificato rispetto al valore precedente

Politica store thru fa una memorizzazione attraversando la cache "come se non ci fosse"e quindi modificando il dato non solo in cache ma anche in Mdl.
Questa politica privilegia la semplicità, le informazioni in Mdl e le loro copie in cache rimangono sempre congruenti.
Non si hanno ulteriori complessità a livello hardware ogni sostituzione di un blocco in cache con un altro blocco non crea nessun problema perche il blocco in cache non contiene nulla di diverso dall' originale.
E una politica non ottimizzata infatti gli accessi in memoria non vengono velocizzati dalla presenza della cache;
E una politica accettabile solo perche gli accessi in scrittura sono decisamente meno do quelli in lettura: per produrre un risultato servono in genere più
operandi;
– ma soprattutto, ci sono tutte le fasi di fetch!

Politica store in che fa una memorizzazione in cache e quindi modifica il dato soltanto in MC.
Privilegia l'ottimizzazione infatti con la politica store in anche gli accessi in scrittura vengono velocizzati se trovano la cella desiderata in cache.
E una politica più complessa perche tra MC e MdL si crea incongruenza: la copia aggiornata è in cache;
• si potrebbe riscrivere il blocco da MC a MdL prima di eliminarlo, ma se i due blocchi sono uguali si perde tempo;
– si introduce un bit di modifica M (per ogni blocco di
MC) che viene settato a ogni scrittura;
– se il blocco da sostituire ha M=1, lo si riscrive in MdL.

Il problema del calcolo del riporto nei circuiti di prodotto , e le possibili soluzioni.

Regola di calcolo
Si costruisce la matrice diagonale dei prodotti parziali:
• dove il moltiplicatore vale 1, si copia il moltiplicando;
• dove il moltiplicatore vale 0, si inseriscono zeri.
Si effettua la somma per colonna dei prodotti parziali. Ad ogni generazione di riporto, si scrive un uno nella colonna immediatamente più significativa a sinistra di quella nella quale si e generata il riporto.
Matrice di calcolo
Ogni elemento della matrice deve calcolare il prodotto parziale:
• bit del moltiplicando AND bit del moltiplicatore.
I bit del moltiplicatore si devono propagare per riga. I bit del moltiplicando si devono propagare in diagonale.
Ogni elemento della matrice deve:
• sommare il prodotto parziale con il risultato parziale della somma in colonna con il risultato proveniente dal elemento soprastante nella stessa colonna della matrice diagonale ;
• tenere conto del riporto in ingresso che può provenire dal elemento della matrice diagonale a destra ;
• generare l'eventuale il riporto in uscita a sinistra per la colonna immediatamente più significativa.
La generazione dei prodotti parziali richiede 1 livello di porte logiche (AND).
Ogni cella introduce ulteriori 2 livelli (circuito FA: FULL ADDER).
Dopo aver completato la prima riga della matrice diagonale (n celle) i riporti devono

discendere lungo la diagonale (n-1 celle).Il numero di livelli totali da attraversare per produrre il risultato è dunque:
NLIVELLI = 1 + 2 × (n + (n-1)) = 4n – 1

Il problema del calcolo del riporto nei circuiti di somma, e le possibili soluzioni

Il circuito capace di sommare 2 numeri a n bit si compone di n FULL ADDER ,ciascun FA riceve i due bit di posto omologo nei due addendi A e B cioè i due bit che sono coefficienti della stessa potenza di due somma questi due bit per generare il bit del risultato sempre coefficiente della stessa potenza di 2 ,riceve in ingresso l'eventuale riporto dalla soma dei bit di peso immediatamente inferiore e genera il riporto per la somma dei bit di peso immediatamente superiore. Il FA di posto 0 potrebbe essere piu correttamente un circuito HA dal momento che non avrà mai riporto di ingresso .
Ogni FA è una rete combinatoria a 2 livelli. Per la somma di 2 numeri a n bit servono n FA. La rete risultante è una rete a 2n livelli (lenta).
Una possibile soluzione alla lentezza del sommatore a n bit e la tecnica dell' c.d carry look ahead .Una rete a 3 livelli per generare direttamente il riporto (i+1)
Infatti e una somma di prodotti quindi 2 livelli ma ogni termine e una funzione generate o propagate e quindi una somma o un prodotto dei bit di dato(A e B) pesati secondo la posizione opportuna Si può dunque anticipare il riporto ("guardare avanti" il riporto) nei limiti della complessità di una rete combinatoria con porte logiche a tanti ingressi: circuiti di CLA usuali: 8 bit o meno.
Un'altra dell' sommatore a n bit e il Sommatore/sottrattore in complemento a 2
Una batteria di porte XOR e un segnale ADD/SUB consentono di effettuare il complemento a 2 "on the fly".
Il circuito risultante esegue sia la somma sia la differenza in complemento a 2

Struttura interna e funzionamento delle diverse componenti presenti in un calcolatore che prevede accessi a disco rigido effettuati mediante DMA.

Il programma in esecuzione da parte della CPU richiede la lettura di un settore (1 Kbyte) da memoria di massa a disco.Il calcolatore in esame è dotato di DMAC per le operazioni di I/O su disco.
Comportamento dei 3 attori coinvolti:
• CPU (attività software 1, hardware 2)
• DMAC (attività 3)
• Interfaccia a disco (attività 4)
1)Il programma in esecuzione chiama la routine readisk del Sistema Operativo;
• La routine readisk inizializza il DMAC:
– inserisce in PA l'identificativo dell'interfaccia;
– inserisce in MDA l'indirizzo della zona di memoria che fa da buffer del disco;
– inserisce in DC il valore 1024 (n° byte da leggere);
– inserisce in TD l'indicazione di lettura (IN).
• La routine readisk inizializza l'interfaccia:
– comunica il numero della traccia e del settore da leggere da disco, indica che si tratta di lettura.
• Il S.O. sospende il programma in esecuzione e lancia altre attività.
4)L'interfaccia segnala dato pronto al DMAC.
3)• Il DMAC chiede i bus alla CPU (HOLDREQ).
2)• La CPU rilascia i bus (HOLDACK).
3)• Il DMAC:
– pone su Address Bus il contenuto di MDA;
– dà il segnale IN all'interfaccia e S alla memoria:
4) un dato viene scritto direttamente da interfaccia a memoria, mediante il Data Bus;
3)– incrementa MDA e decrementa DC;
3)– toglie la richiesta dei bus (HOLDREQ).
2)• La CPU disattiva HOLDACK e si riappropria dei bus.
3)Il DMAC genera una interruzione.
2)• La CPU riconosce l'interruzione e attiva la routine di risposta all'interrupt di DMAC.
1)• La routine di risposta all'interrupt segnala a readisk che l'operazione è finita (per es. forzando a TRUE un opportuno flag).
1)• Il S.O. riattiva il programma che aveva richiesto la lettura del settore da disco, e che a questo punto può riprendere le proprie attività.

Quando e perché ha senso ricorrere alla tecnica del DMA nelle operazioni di Input/Output di un calcolatore?

Sono spesso presenti nel calcolatore fenomeni di I/O che si ripetono ad alta frequenza:
• trasferimento di settori da/verso memoria di massa;
• trasmissione/ricezione di frames da rete.
Questi fenomeni richiedono generalmente il trasferimento da periferica a memoria o viceversa di sequenze di dati (celle). Solo quando il trasferimento dell'intera sequenza è terminato, si può procedere con l'elaborazione. Anche qualora sia dedicata a questo tipo di operazioni di I/O, la CPU è penalizzata perché:
• essendo un componente general purpose, deve scoprire (fetch di istruzioni macchina) passo passo cosa le si chiede di fare;

• per trasferire un dato da periferica a memoria o viceversa, il singolo accesso utile (trasferimento del dato) è penalizzato da un elevato numero di accessi "inutili":
– fetch delle istruzioni;
– incremento del puntatore all'area di memoria da/in cui trasferire i dati;
– aggiornamento del contatore di dati trasferiti...
Proprio per evitare che sia la CPU a doversi ocupare del transferimento di ogni singolo byte e stata messa a punto la tecnica DMA(Direct Memory Access). Questa tecnica prevede la possibilità che altri dispositivi - oltre alla CPU - possano accedere a memoria:
• diventare quindi temporaneamente Master del bus.
Per far questo, serve poter richiedere alla CPU la possibilità di utilizzare il bus:
• viene inserita una linea dedicata del bus di controllo: HOLDREQ.
Anche gli stadi di uscita della CPU che pilotano le linee dell'Address Bus e del Control Bus devono essere TRI STATE (per lasciare agli altri Master la possibilità di pilotarle). In un calcolatore possono esistere più periferiche che richiedono DMA.
Non c'è possibilità di sincronizzazione fra le richieste di DMA:
• ogni gestore di DMA chiede i bus quando la propria periferica deve trasferire un dato.
La linea di richiesta HOLDREQ deve essere gestita mediante porte OPEN COLLECTOR:
• linea attiva bassa;
• chi vuole i bus forza a 0 a bassa impedenza la linea;
• normalmente la linea è tenuta a 1 dalla resistenza di pull-up.
DMAC (DMA Controller) -Integrato di supporto alla gestione del DMA.
La CPU programma il DMAC comunicando:
• indirizzo della zona di memoria da/in cui trasferire i dati;
• numero di dati da trasferire;
• identificativo della periferica e senso di trasferimento.
Quando la periferica segnala di essere pronta:
• il DMAC richiede i bus con il segnale HOLDREQ;
• quando la CPU ne ha terminato l'eventuale uso in corso, ne segnala il rilascio con HOLDACK;
• il DMAC effettua il trasferimento e aggiorna puntatori e contatori;
• finito l'intero trasferimento, genera interrupt.