

1. Che cosa si intende per interruzione. Cosa sono e come vengono gestite le interruzioni multiple?

Un **interrupt** è un tipo particolare di istruzione della CPU che consente l'interruzione di un processo qualora si verificano determinate condizioni oppure il processo in esecuzione debba effettuare una richiesta al sistema operativo. Evento generato internamente (overflow) o esternamente (I/O) che trasferisce il controllo ad un programma specifico in risposta all'evento.

Viene utilizzata per :

Errori: un processo tenta di eseguire un'istruzione non valida, come una divisione per zero. In questi casi non è possibile proseguire con l'esecuzione del processo e l'interrupt consente di informare il sistema operativo di quanto avvenuto in modo da permettere la corretta gestione del problema.

Gestione I/O: un processo richiede un'operazione di I/O al sistema operativo. Le CPU moderne prevedono la possibilità di utilizzare diversi livelli di privilegi che i processi in esecuzione hanno, per ragioni di sicurezza. Solo il sistema operativo può effettuare alcune operazioni, accedere ad alcune aree di memoria, gestire le periferiche.

I/O: un dispositivo di I/O informa la CPU che è disponibile a ricevere o fornire dati. In questo caso viene avviata un'opportuna procedura del sistema operativo preposta ad occuparsi della relativa periferica. Questo tipo di interrupt necessita una gestione molto attenta da parte della CPU. Infatti è possibile che due dispositivi generino contemporaneamente un'interrupt, ed è necessario disporre di meccanismi che evitino conflitti e la perdita di informazioni, ad esempio decidendo quale interrupt ha maggiore priorità e deve essere eseguito per primo e ponendo in coda il secondo.

La gestione delle **Interruzioni Multiple** avviene mediante il meccanismo di annidamento. Gli indirizzi di ritorno e i relativi contesti vengono memorizzati nello stack. Viene introdotto un meccanismo di priorità: viene eseguita solo l'INT con priorità maggiore di quella che si sta eseguendo.

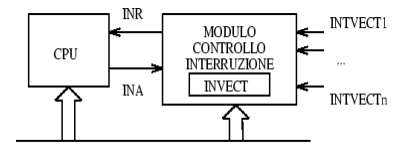
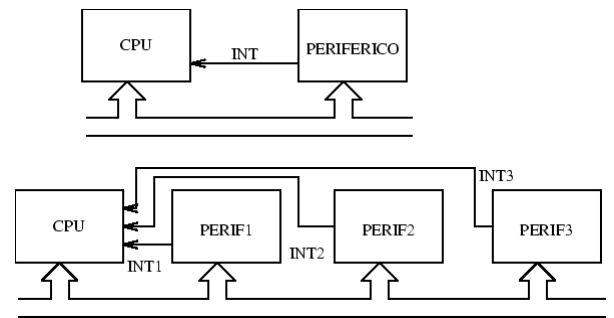
Gestione Interrupt: la presenza di un interrupt è segnalata dalla CPU da una linea proveniente dall'esterno. Il segnale viene memorizzato e testato dalla CPU alla fine di ogni ciclo di istruzione. La CPU risponde trasferendo il controllo a un altro programma. L'evento che causa l'interruzione è asincrono rispetto all'esecuzione del programma.

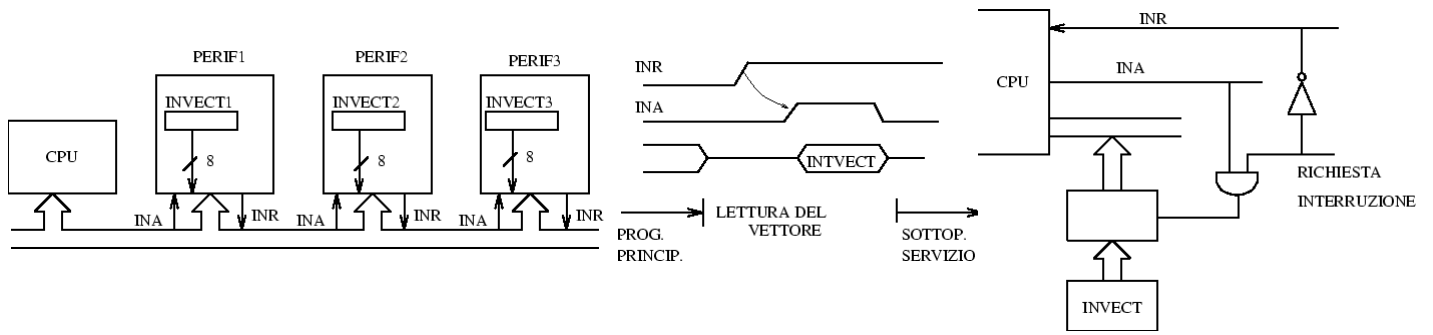
Azioni della CPU in risposta alla richiesta di interruzione:

- Identificazione della sorgente di interruzione: A cura della CPU (polling) e a cura dell'elemento che genera l'interruzione.
- La CPU ottiene l'indirizzo di memoria del programma di servizio: al termine della fase di polling e durante la stessa richiesta di interruzione.
- Il PC e altre informazioni essenziali relative alla CPU vengono memorizzate automaticamente.
- Lo stato complessivo della CPU viene memorizzato a cura del programma di gestione.
- Nel PC viene immagazzinato l'indirizzo del sottoprogramma di gestione.
- L'esecuzione del sottoprogramma continua fino all'istruzione di RETURN che riporta il controllo al programma interrotto.

Gestione centralizzata: tutte le richieste sono gestite da un solo modulo centralizzato.

Gestione distribuita: la selezione del periferico avviene con **Polling** (ogni elemento periferico dispone di un registro che contiene un bit che segnala la richiesta di interruzione; una sola routine legge in sequenza tutti i registri) o **Vettorizzazione** (ogni periferico invia alla CPU un valore o vettore identificatore; il vettore viene utilizzato per individuare il programma opportuno di gestione dell'interruzione).





Generazione vettore interruzione

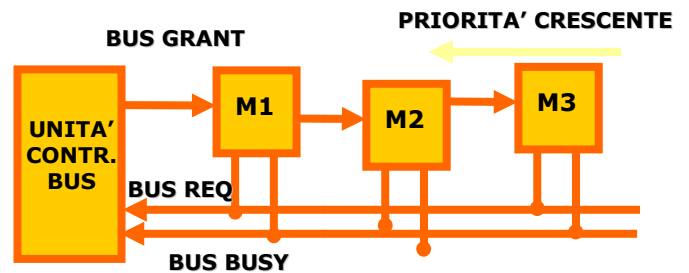
2. Arbitri concentrati e distribuiti per la gestione degli accessi ad un bus.

Se si hanno più master collegati ad un bus serve un arbitro che decida quale unità ha il controllo. Due tipi: centralizzato (possibilità di modificare agevolmente la politica di assegnazione, maggiore osservabilità), distribuito (maggiore modularità e possibilità di espansione e riconfigurazione, migliore tolleranza ai guasti).

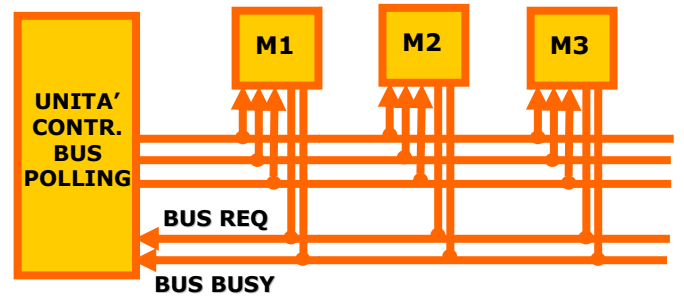
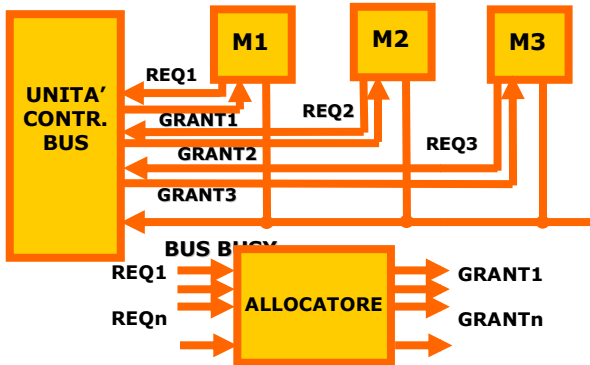
Tecniche di arbitrato (a distribuzione decrescente):

A catena o Daisy Chain: i moduli sono collegati fisicamente a catena. La posizione fisica è associata alla priorità, la quale non è facilmente modificabile.

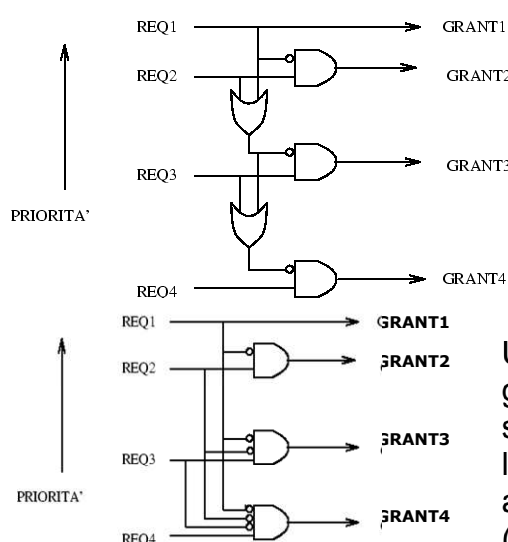
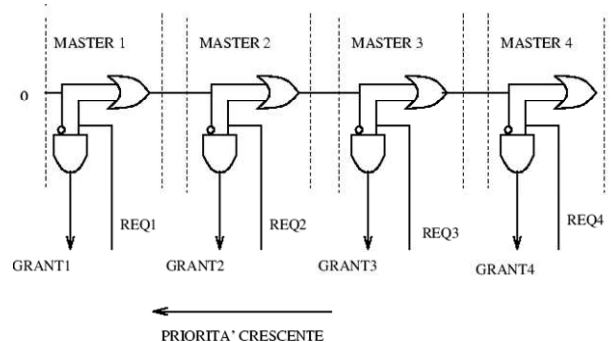
Polling: in corrispondenza di una richiesta il modulo di controllo genera una sequenza di indirizzi corrispondenti alle varie unità. Il modulo indirizzato con una richiesta pendente assume il controllo.



ARBITRO CONCENTRATO



ARBITRO DISTRIBUITO



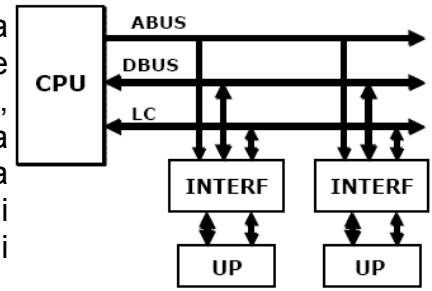
Due possibili versioni dell'allocatore concentrato

Il circuito dell'arbitro distribuito è eguale a quello concentrato, varia la distribuzione delle parti hardware.

Un meccanismo di arbitrato presuppone delle politiche di gestione della **priorità**. Le politiche di priorità possono essere statiche o dinamiche. Le politiche di priorità devono consentire l'esecuzione di tutte le attività (**fairness**), impedire che elementi a bassa priorità non possano mai avere accesso alla risorsa (**starvation**).

3. Gestione di I/O a controllo di programma.

Tecnica più immediata per gestione I/O, il processore si limita a eseguire appositi sottoprogrammi di "ingresso" o "uscita" durante l'esecuzione di un programma. Con questa tecnica il processore, prima di effettuare l'operazione di I/O controlla che la periferica sia pronta al trasferimento, e se ciò non si verifica, rimane in attesa finché dalla periferica stessa non giunge l'opportuno segnale di disponibilità. Questi cicli di attesa permettono di evitare di perdere i dati inviati ad una periferica che in quel momento è occupata.




Possiamo utilizzare un flag di disponibilità; forzato a 1 dalla periferica, quando questa è disponibile a ricevere/inviare un dato, e forzato a 0 dal processore durante l'operazione di trasferimento dati. L'interfaccia deve fornire: un registro per i dati trasferiti, un registro per i comandi alla periferica, un registro per lo stato della periferica. Questo metodo di gestione penalizza la velocità di elaborazione perché il processore entra in un ciclo di attesa inoperosa, per contro richiede però interfacce molto semplici e consente di effettuare direttamente da programma un controllo sull' I/O.

4. Le topologie e le caratteristiche delle strutture di interconnessione dei sistemi paralleli.

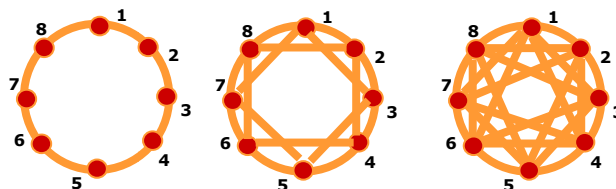
La struttura di interconnessione è un elemento essenziale in un sistema di elaborazione distribuito tanto da condizionarne le prestazioni complessive in modo talvolta rilevante. A questo livello di astrazione un sistema distribuito è formato da nodi (di elaborazione) e da rami per il trasporto dell'informazione. Le caratteristiche principali sono: la topologia della rete, la modularità e riconfigurabilità, le prestazioni in termini di banda passante, la latenza. Possono essere statiche o dinamiche. Statiche: la topologia di interconnessione formata da archi punto a punto non cambia durante l'esecuzione di un programma. Dinamiche: canali commutati consentono alla struttura di comunicazione di adattarsi alle esigenze specifiche del programma. Dal grado del nodo (numero dei rami che fanno capo al singolo nodo) di un sistema dipendono modularità e costo perché se ogni nodo è una unità di elaborazione la presenza di più rami connessi richiede altrettante porte di I/O. Il diametro della rete (massimo dei cammini minimi tra due nodi della rete) viene associato alla latenza del processo di comunicazione; influisce sul ritardo massimo presentato dalle informazioni (messaggi) che circolano sulla rete.

Le tecniche di instradamento dei pacchetti all'interno di una rete di interconnessione sono di due tipi: commutazione di pacchetto, commutazione di circuito. Commutazione di pacchetto: Store & Forward: consiste nell'immagazzinare (store) completamente il messaggio prima di spedirlo (forward) al nodo successivo. Wormhole: consiste nell'inviare immediatamente al nodo successivo le parti di messaggio già ricevute; molto più performante del store&forward, ma la complessità della rete aumenta.

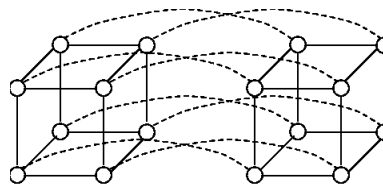
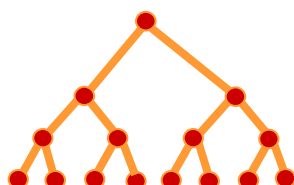
Commutazione di circuito: si stabilisce un collegamento fisico tra i due nodi coinvolti nello scambio di informazioni.

Array lineare: rete monodimensionale con N nodi connessi da N-1 canali (non è un bus !). Grado 2, diametro N-1. 

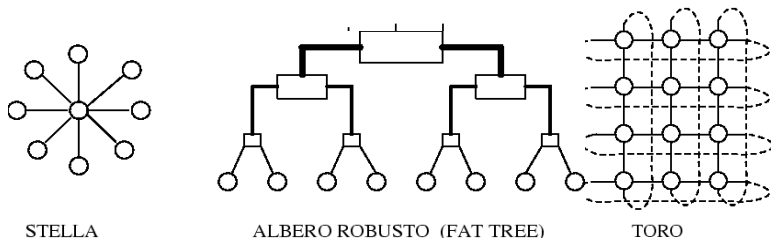
Anello: Da una rete monodimensionale con l'aggiunta di un collegamento. Grado 2, diametro N/2 (bidirezionale) o N (monodirezionale).



Albero: struttura gerarchica a livelli.



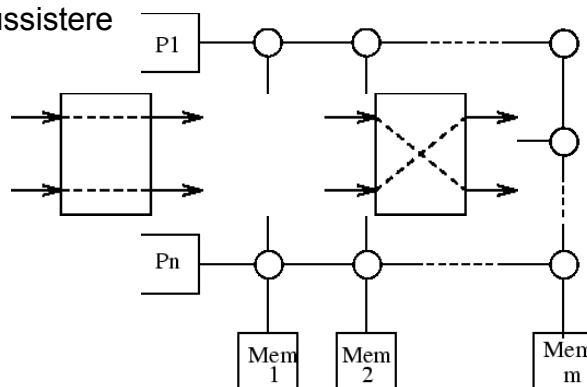
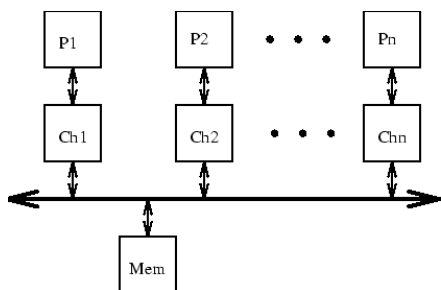
Altre strutture:



Reti dinamiche: bus, cross-bar, reti multistadio.

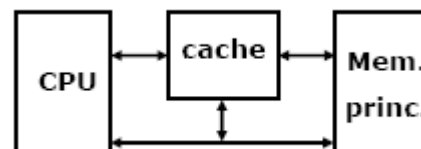
Cross-bar: assenza di contesa se i processori accedono a memoria diversa. Alto costo del nodo di interconnessione.

Reti multistadio: un elemento di commutazione che può sussistere in due stati diversi.

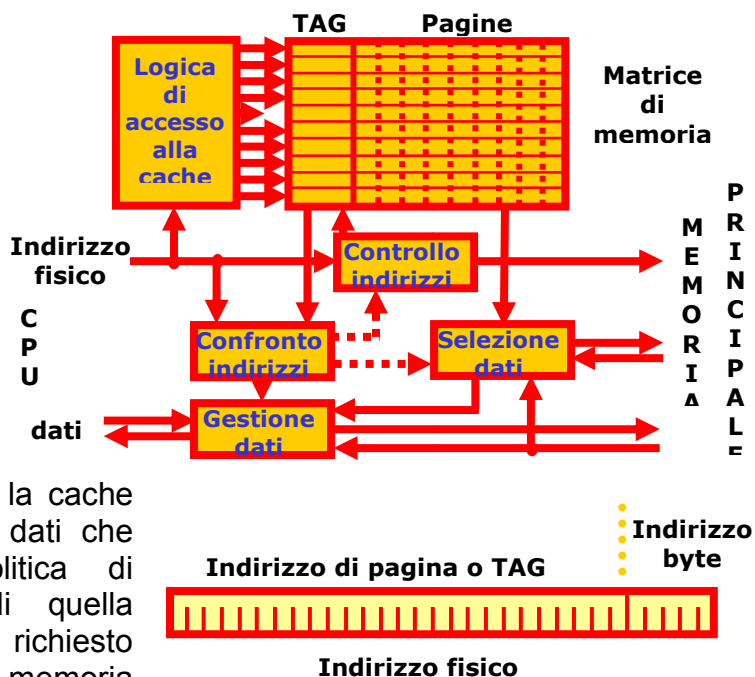


5. Descrivere i meccanismi di funzionamento di una memoria cache. Tecniche di accesso.

La memoria interna è composta da registri interni alla CPU, visibili o no dal programmatore, di dimensioni contenute e bassi tempi di accesso, in cui si memorizzano temporaneamente dati e istruzioni. La cache è una memoria veloce e di piccole dimensioni posta fra la CPU e la memoria principale. La cache e la memoria principale formano una gerarchia di memoria. Le prestazioni della memoria cache dipendono anche dalla sua posizione rispetto alla CPU: cache su scheda, cache su chip. La presenza di una memoria cache sullo stesso chip del processore rappresenta la soluzione che garantisce la maggiore efficienza. Nei sistemi più recenti sono presenti entrambe le soluzioni. I dati vengono memorizzati in pagine di cache (o linee) caratterizzate da una parte dell'indirizzo o tag. Ciclo operazioni: la cache confronta la parte rilevante dell'indirizzo, se vi è {hit} viene completato il ciclo. In caso di {miss} la cache inizia la lettura da memoria principale di dati che comprendono quello richiesto. La politica di sostituzione ha gli stessi problemi di quella preemptive della memoria virtuale. Il dato richiesto deve essere trovato rapidamente nella memoria cache (tecniche associative, accesso diretto, tecniche miste set-associative).



Tecniche associative: la cache è organizzata come una memoria associativa (costosa) e il tag viene utilizzato come chiave di lettura. Accesso diretto: la memoria cache M_1 è suddivisa in $S_1 = 2^k$ pagine o linee di n parole. La memoria principale M_2 è suddivisa in S_2 regioni di uguali dimensioni. $M1(1), \dots, M1(i), \dots, M1(S_1)$: pagine in cache. $M2(1), \dots, M2(j), \dots, M2(S_2)$: pagine in memoria principale. $M2(j)$ va in $M1(i)$ con $i = j \pmod{S_1}$. Funziona rapidamente perché è noto in che indirizzo della cache la mia pagina potrebbe essere ricaricata. Tecnica set-associative: le linee nella cache sono suddivise in gruppi. Ogni gruppo può ospitare linee della memoria principale individuate secondo la tecnica di accesso diretto. All'interno di ogni



Tecniche associative: la cache è organizzata come una memoria associativa (costosa) e il tag viene utilizzato come chiave di lettura.

Accesso diretto: la memoria cache M_1 è suddivisa in $S_1 = 2^k$ pagine o linee di n parole. La memoria principale M_2 è suddivisa in S_2 regioni di uguali dimensioni. $M1(1), \dots, M1(i), \dots, M1(S_1)$: pagine in cache. $M2(1), \dots, M2(j), \dots, M2(S_2)$: pagine in memoria principale. $M2(j)$ va in $M1(i)$ con $i = j \pmod{S_1}$. Funziona rapidamente perché è noto in che indirizzo della cache la mia pagina potrebbe essere ricaricata.

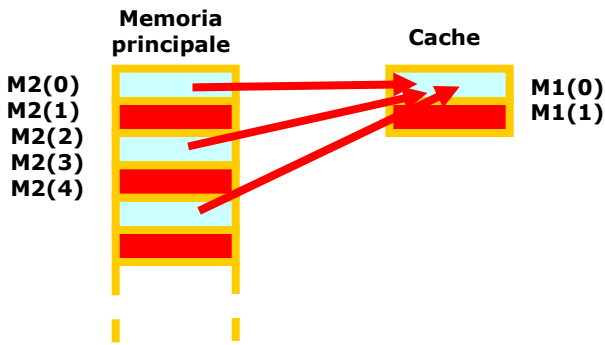
Tecnica set-associative: le linee nella cache sono suddivise in gruppi. Ogni gruppo può ospitare linee della memoria principale individuate secondo la tecnica di accesso diretto. All'interno di ogni

gruppo la linea viene individuata in modo associativo. La tecnica associativa è molto costosa quando ha dimensioni elevate, in questo modo viene spezzata in tante piccole parti.

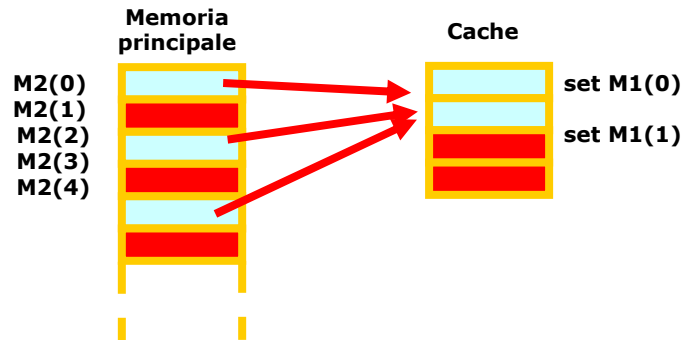
Quando la CPU scrive sulla memoria cache si genera una discrepanza fra il valore del dato nella memoria principale e nella cache. Con *write-back* (o *copy-back*), quando viene modificato un dato nella cache la linea corrispondente viene marcata. Quando una linea è scaricata viene ricopiata nella memoria principale solo se marcata. Con *write-through* (o *copy-through*), quando viene modificato un dato nella cache, lo stesso viene modificato anche nella memoria principale. In un programma il rapporto fra le operazioni di lettura e di scrittura è dell'ordine di 10.

Alternative: cache separate per i dati e per le istruzioni. Livelli di cache multipli.

ACCESSO DIRETTO



ACCESSO DIRETTO



6. Critiche al parallelismo: la legge di Amdahl

Le critiche al parallelismo derivano dal fatto che molti problemi sono espressi in termini sequenziali ed esistono numerosi programmi già sviluppati in maniera sequenziale che sarebbe conveniente eseguire più velocemente. Esistono quindi ragioni non tecniche nell'introduzione e nell'utilizzo efficace dei sistemi di calcolo paralleli per ottenere prestazioni superiori a quelle delle architetture sequenziali. In particolare esistono già molti programmi sviluppati per architetture sequenziali e le conoscenze dei programmatori e la loro abitudine a sviluppare programmi su architetture sequenziali rappresentano un investimento che non può essere trascurato. Lo sfruttamento efficace di tutte le caratteristiche delle architetture parallele richiede conoscenze che chi sviluppa le applicazioni spesso non ha.

La legge di Amdahl riassume i problemi dovuti alla presenza di operazioni sequenziali all'interno di un programma che deve essere eseguito su sistemi paralleli. In particolare se indichiamo con:

- P: Numero di processori
- T(P): tempo di esecuzione di un programma a parallelismo P.
- f: frazione di programma che deve essere eseguita sequenzialmente

Tempo di esecuzione del programma su di una architettura a parallelismo P: $T(P) = f T(1) + (1-f) [T(1)/P]$

Speed Up: $S(P) = T(1)/T(P) = 1 / [f + (1-f)/P] = P / [1 + f(P-1)]$.

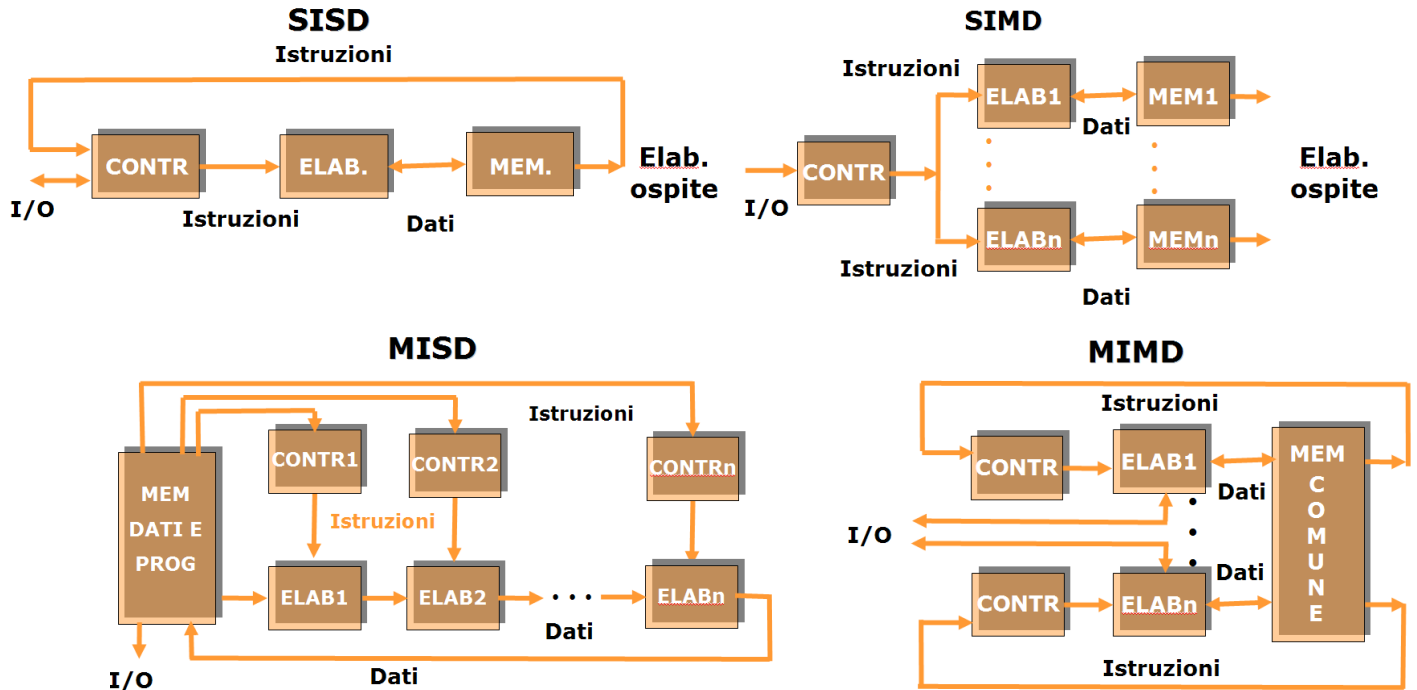
Efficienza: $E(P) = S(P)/P = 1 / [1 + f(P-1)]$.

Dall'analisi dei risultati ottenuti si deduce che una piccola porzione di codice non parallelizzabile può avere un grande effetto globale.

Riuscire a parallelizzare una porzione di codice sequenziale consente di migliorare significativamente le prestazioni.

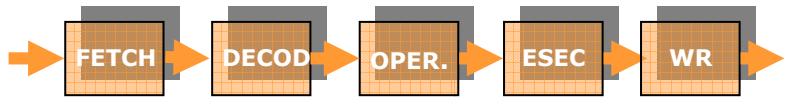
7. Classificazione di Flynn per le architetture parallele.

Un sistema di elaborazione opera su un flusso di dati in base ad un flusso di istruzioni acquisite dalla memoria. Flynn classifica i sistemi di calcolo in base al grado di parallelismo di questi due flussi. SISD (Singolo flusso istruzioni, singolo flusso dati): architettura tradizionale con singola cpu che elabora una istruzione alla volta operando su un dato alla volta, SIMD (Singolo flusso istruzioni, flusso di dati multiplo): più cpu operano in modo sincrono eseguendo la stessa istruzione su dati diversi, MISD (Flusso di istruzioni multiplo, singolo flusso dati): il medesimo flusso di dati viene elaborato da un insieme di processori che eseguono istruzioni diverse, e MIMD (Flusso di istruzioni multiplo, flusso di dati multiplo): unità di elaborazione diverse eseguono istruzioni diverse su dati diversi.



8. Indicare le principali caratteristiche delle architetture pipeline e i vantaggi ottenuti dalla loro introduzione. Indicare le funzioni svolte dai singoli stadi.

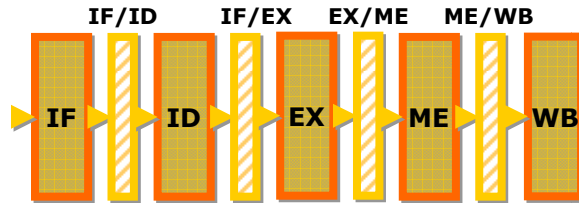
Nelle architetture pipeline l'esecuzione di una singola istruzione consiste di più fasi, svolte in sequenza da sezioni diverse. In questo modo si aumenta il throughput senza aumentare la velocità dell'unità centrale. Si introduce tuttavia un ritardo tra ingresso e uscita dei dati.



	T ₁	T ₂	T ₃	T ₄	T ₅					
i	IF	ID	EX	ME	WB					
i+1		IF	ID	EX	ME	WB				
i+2			IF	ID	EX	ME	WB			
i+3				IF	ID	EX	ME	WB		
i+4					IF	ID	EX	ME	WB	
i+5						IF	ID	EX	ME	WB

Le fasi sono: fetch dell'istruzione dalla memoria (FETCH), decodifica dell'istruzione (DECOD), cattura dell'operando dalla memoria (OPER), esecuzione dell'istruzione (ESEC), scrittura dei dati (WR). Il numero di fasi nelle quali è ripartito lo svolgimento della singola istruzione è una scelta di progetto.

A regime (pipeline piena) si ha un incremento prestazionale: ogni istruzione richiede k periodi di clock, ma ad ogni periodo viene completata un'istruzione. Rispetto all'esecuzione in sequenza le volte superiori, ma non è sempre la pipeline piena. Rispetto a una pipeline lineare a k stadi in: $T_k = [k + (n-1)]\tau$ elabora n istruzioni.



viene completata un'istruzione. all'esecuzione in sequenza le volte superiori, ma non è sempre la pipeline piena.

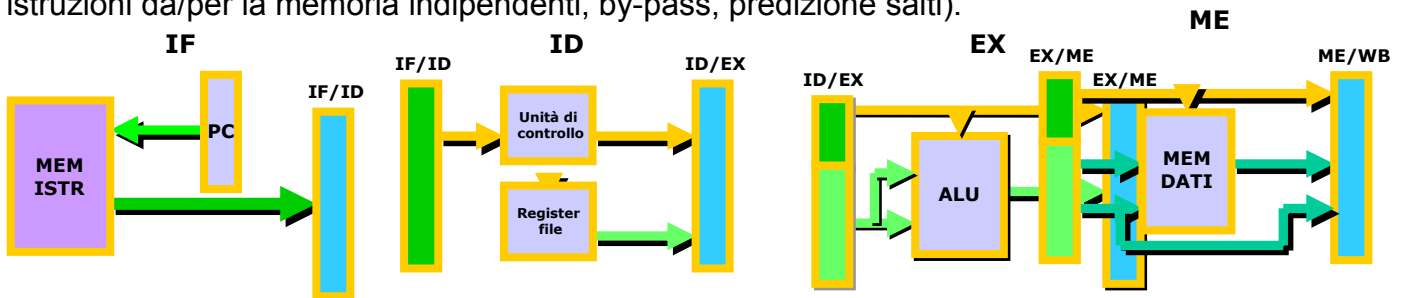
Una pipeline lineare a k stadi in: $T_k = [k + (n-1)]\tau$

Speed-up:

$$S_k = T_1 / T_k = nk\tau / [k\tau + (n-1)\tau]$$

I limiti risiedono nel costo ($2 < k < 15$), dalla dipendenza dei dati, salti, interruzioni, ecc..

Cinque stadi: IF (fetch o prelievo dell'istruzione), ID (decodifica e lettura dei registri), EX (esecuzione), ME (accesso alla memoria), WB (scrittura registro destinazione). Il periodo di clock viene scelto in modo da permettere la propagazione completa dei segnali. Si può incappare in conflitti strutturali, dati e di controllo. Conflitti strutturali: due stadi tentano di accedere alla stessa memoria. Si può pensare a due cache separate per i dati e le istruzioni. Per evitare gli stalli ci sono diverse possibilità: soluzioni software (riordinamento del codice), soluzioni hardware (percorsi dati e istruzioni da/per la memoria indipendenti, by-pass, predizione salti).

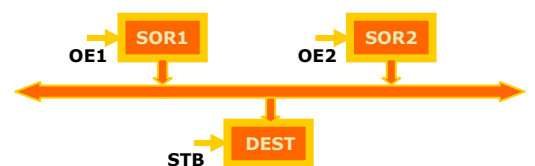


9. Sistemi a bus. Protocolli per il trasferimento dei dati e le operazioni di lettura e scrittura.

Sistema (combinazione del trasferimento a collettore e a diffusione) in cui più moduli possono trasferirsi dati utilizzando una sola struttura di interconnessione, il bus. Il bus è un canale sincrono attraverso cui diversi componenti elettronici (quali ad esempio le varie parti di un computer) dialogano fra loro. I trasferimenti sono gestiti dalla combinazione dei segnali di abilitazione OE e acquisizione STB. Esiste inoltre un modulo di controllo che

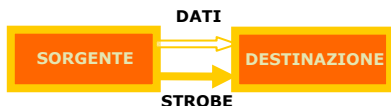
invia i segnali STBi e OEj. È possibile raggruppare gli elementi qui considerati in moduli. Modulo MASTER (CPU): controllo + sorgenti e/o destinazioni, modulo SLAVE: sorgenti e/o destinazioni. Lettura: trasferimento dati da SLAVE (sorgente) a MASTER (destinazione). Scrittura: trasferimento dati da MASTER (sorgente) a SLAVE (destinazione). I moduli giocano nelle due operazioni ruoli diversi; in ogni caso però è sempre il modulo MASTER che attiva l'operazione e sceglie il modulo SLAVE.

TRASFERIMENTO A COLLETTORE



Più elementi possono inviare dati a un destinatario.

TRASFERIMENTO DATI PUNTO A PUNTO



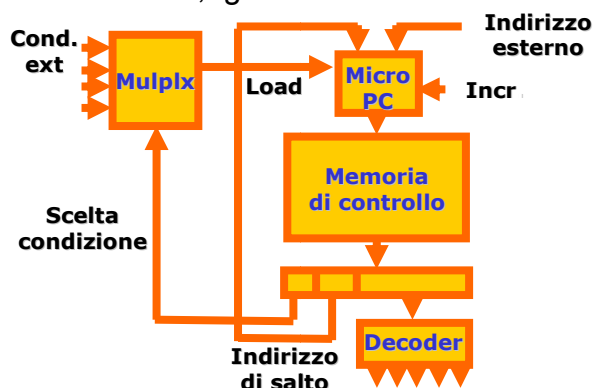
Il segnale STROBE comanda la memorizzazione del dato. Il trasferimento punto a punto richiede un solo segnale di STROBE per segnalare all'elemento ricevente la presenza del dato corretto.

TRASFERIMENTO A DIFFUSIONE



10. Architettura e modo di funzionamento di una unità di controllo microprogrammata.

Nella microprogrammazione le sequenze di comandi emessi a ogni ciclo possono essere considerate come un insieme di istruzioni elementari a un livello di complessità inferiore rispetto alla istruzioni macchina (microistruzioni). L'unità di controllo deve assumere la struttura di un calcolatore a cui viene demandata l'esecuzione del microprogramma associato all'istruzione da eseguire. Ogni istruzione CPU è composta da una serie di microistruzioni, ogni microistruzione realizza microoperazioni concorrenti; ogni microistruzione attiva le microoperazioni con un insieme di linee di controllo. Nell'unità possiamo distinguere: memoria di controllo (ROM PROM o RAM che contiene tutti i microprogrammi), un contatore di microprogramma (funzionalità del Program Counter), un registro di microistruzione corrente. Per compiere un'istruzione: 1) il codice operativo contenuto nel registro istruzioni viene inviato a un decodificatore di istruzioni che genera un indirizzo di microprogramma; 2) questo indirizzo viene caricato nel microPC e punta nella memoria di controllo alla prima microistruzione del microprogramma associato all'istruzione; 3) le microistruzioni vengono eseguite in sequenza e ad ogni passo i segnali corrispondenti vanno ad abilitare le rispettive unità funzionali e nel microPC viene messo l'indirizzo della microistruzione successiva. L'organizzazione microprogrammata consente al progettista di sistemi di estendere o modificare a sua discrezione il repertorio delle istruzioni del processore senza intervenire sull'hardware, visto che per modificare le istruzioni basterà scrivere dei nuovi microprogrammi ed implementarli su una nuova memoria.



11. Tassonomie delle architetture distribuite

Le architetture parallele comprendono: architetture pipeline, architetture superscalari, architetture vettoriali, array processors, multiprocessori.

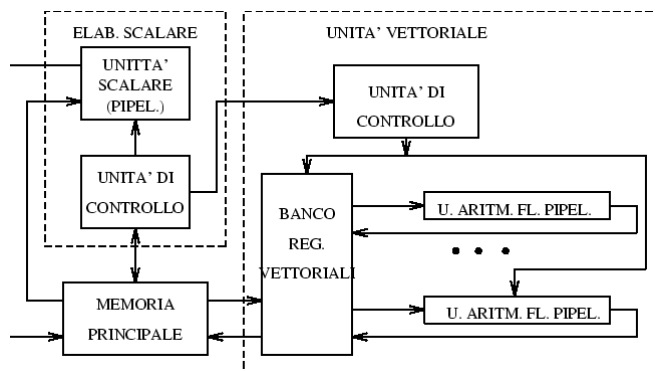
Architetture pipeline: l'esecuzione di una singola istruzione consiste di più fasi, svolte in sequenza da sezioni diverse. In questo modo si aumenta il throughput senza aumentare la velocità dell'unità centrale. Si introduce tuttavia un ritardo tra ingresso e uscita dei dati.

Le fasi sono: fetch dell'istruzione dalla memoria (FETCH), decodifica dell'istruzione (DECOD), cattura dell'operando dalla memoria (OPER), esecuzione dell'istruzione (ESEC), scrittura dei dati (WR).

Architetture superscalari: parlando per esempio di un supercalcolatore, si ha un buon rapporto di prestazioni fra elaboratore scalare e vettoriale. Il sistema è maggiormente scalabile con l'aumento del numero di processori, ha grande capacità di memoria e buone capacità di I/O.

Architetture vettoriali: composte da un insieme di unità funzionali (ALU Floating Point Pipelined), di registri per dati vettoriali e di un'unità di esecuzione scalare associata. I vantaggi associati a questa architettura sono l'esecuzione in parallelo di operazioni più complesse, la diminuzione di conflitti per l'accesso alla memoria e la possibilità di usare codici sequenziali con compilatori specializzati.

Array Processors: strutture con più processori connessi in modo regolare. Le unità di elaborazione vengono aggiunte ad un calcolatore ospite tradizionale. È una struttura adatta alla tecnologia VLSI



(elementi uguali, connessione fra vicini, velocità limitata delle operazioni). Esistono gli array SIMD, MIMD e sistolici.

Multiprocessori: unità di elaborazione tradizionali connesse da una rete di interconnessione. Un esempio è costituito dai cluster, classe di architetture di elaborazione parallele basate su un insieme di elaboratori indipendenti cooperante per mezzo di una rete di interconnessione e coordinato mediante un sistema operativo che lo rende in grado di operare su di un singolo workload.

Un commodity cluster è costituito invece da nodi di elaborazione commerciali (COTS –Commercial off the shelf) in grado di operare in modo indipendente e collegati tramite una rete di interconnessione (LAN o SAN) anch'essa di tipo COTS.

12. Architetture superscalari. Mostrare un esempio di architettura superscalare e discuterne i vantaggi e i problemi introdotti (ad es. esecuzione fuori ordine).

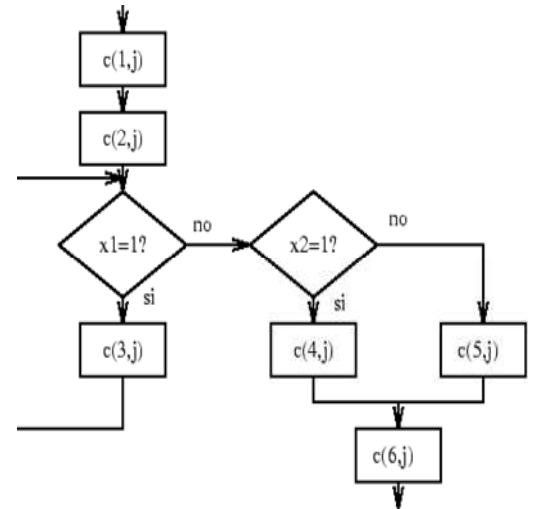
Un microprocessore con architettura superscalare supporta il calcolo parallelo su un singolo chip, permettendo prestazioni molto superiori a parità di clock rispetto ad una CPU ordinaria. In processore superscalare istruzioni diverse trattano i propri operandi contemporaneamente, su diverse unità hardware all'interno dello stesso chip. In questo modo più istruzioni possono essere eseguite nello stesso ciclo di clock. In una CPU superscalare sono presenti diverse unità funzionali dello stesso tipo, con dispositivi aggiuntivi per distribuire le istruzioni alle varie unità. Per esempio, sono generalmente presenti unità per il calcolo intero (AI), floating (AF), branch (B) e load/store (L/S). Si può creare parallelismo adottando più unità funzionali per non far pagare anche alle istruzioni più veloci il ritardo nella fase di EX (IF→ID→EX→ME→WB). Le operazioni AF, ad esempio, richiederanno sicuramente più cicli di clock rispetto alle operazioni AI (come del resto le operazioni di L/S) quindi da qui nasce il problema dell'ordine di completamento non mantenuto; infatti se in ingresso ho una operazione tra float e poi un'operazione tra interi, in uscita avrò prima il risultato della seconda e poi quello della prima. Il compilatore può cercare di riordinare le istruzioni in modo da ottimizzare l'uso delle entità ma in generale è meglio che la CPU faccia concludere le istruzioni in ordine.

13. Unità di controllo cablate

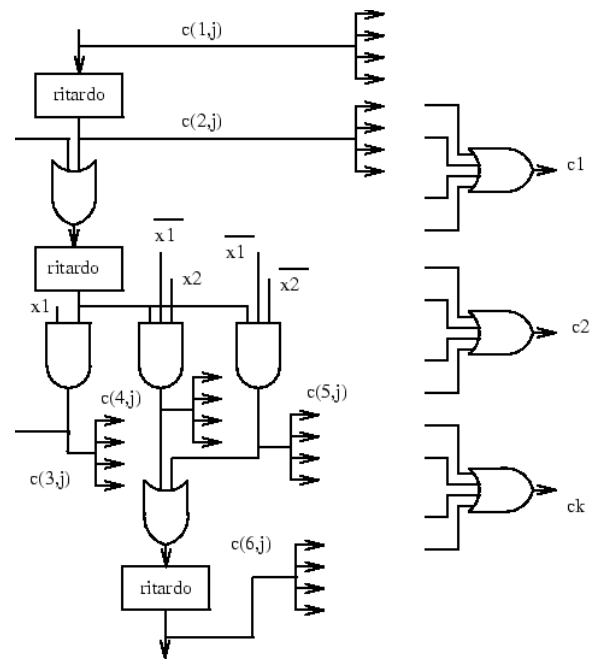
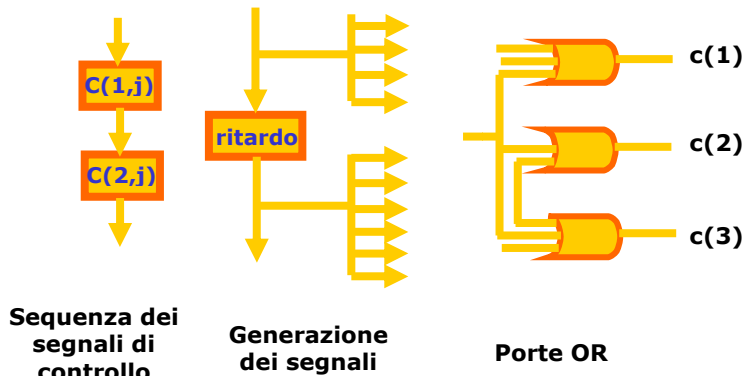
Le unità di controllo cablate sono circuiti sequenziali speciali che realizzano una determinata funzione desiderata. Vantaggi: la loro realizzazione è meno costosa per sistemi semplici, più veloce, più ottimizzata.

Svantaggi: sono di difficile progettazione per sistemi complessi, scarsa flessibilità, costi alti per la correzione di eventuali errori. Esistono due metodologie di progetto: macchine a stati finiti (progetto tradizionale di reti sequenziali, rete combinatoria più registri di stato), generatori di sequenze (basati su elementi di ritardo, asincroni, o contatori, sincroni).

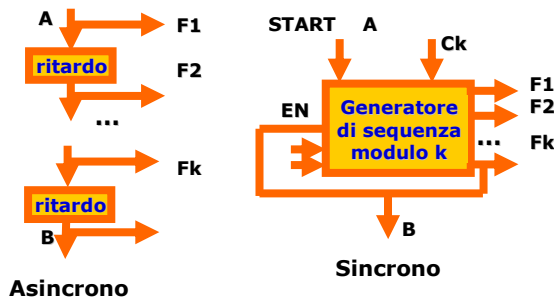
La funzione svolta è quella di generare una sequenza temporale di vettori a partire da un diagramma di flusso (o un programma) che definisce il comportamento della parte operativa.



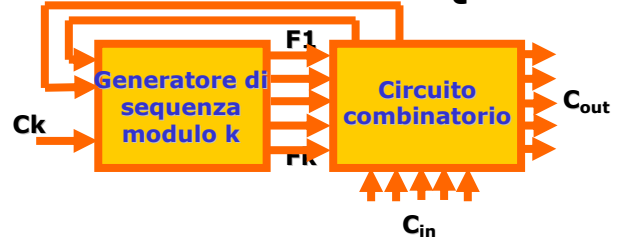
PROGETTO BASATO SU ELEMENTI DI RITARDO



PROGETTO BASATO SU CONTATORI
 La sequenza di segnali di comando $c(ij)$ è generata a partire da una sequenza di segnali di fase F_n



UNITA' DI CONTROLLO BASATA SU CONTATORE DI SEQUENZA



14. I processori CISC.

CISC è l'acronimo di Complex Instruction Set Computer: tipicamente un processore di questo tipo implementa un numero relativamente scarso (una decina) di registri di uso generale, ed ha una unità di controllo microprogrammata: vale a dire che ogni opcode viene tradotto in una serie di passi da compiere secondo un procedimento software interno all'unità di controllo stessa, che è in pratica una CPU nella CPU.

Il set di istruzioni associato a CPU di tipo CISC è molto esteso e composto in genere di alcune centinaia di codici operativi diversi che svolgono funzioni anche molto complesse, fra cui sono caratteristici i trasferimenti memoria-memoria, assenti nei RISC; le istruzioni hanno lunghezza variabile e possono presentarsi in formati diversi, e sono necessari due o più (a volte molti di più) cicli di clock per completare una istruzione; è possibile specificare la posizione dei dati necessari alle istruzioni usando molti metodi di indirizzamento diversi. Il ridotto numero di registri interni obbliga questi processori a scrivere in memoria ogni volta che si verifica una chiamata di funzione, che si verifica un context switch o che viene salvato un registro nello stack.

Programmare in Assembler un CPU CISC è un compito (relativamente) facile, perché le istruzioni presenti sono più vicine a quelle dei linguaggi ad alto livello: i compilatori che generano codice per CPU CISC sono piuttosto semplici ed inoltre il codice prodotto è molto compatto e occupa poca memoria.

15. I processori RISC.

RISC è l'acronimo di Reduced Instruction Set Computer: questi processori hanno una unità di controllo cablata molto semplice e riservano invece molto spazio per i registri interni: una CPU RISC ha di solito da un minimo di un centinaio ad alcune migliaia di registri interni generici, organizzati in un file di registri. Il tipico set di istruzioni RISC è molto piccolo, circa sessanta o settanta istruzioni molto elementari (logiche, aritmetiche e istruzioni di trasferimento memoria-registro e registro-registro): hanno tutte lo stesso formato e la stessa lunghezza, e tutte o quasi vengono eseguite in un solo ciclo di clock. Sono presenti solo un numero ristretto di metodi di indirizzamento. Il fatto di avere un formato unico di istruzione permette di strutturare l'unità di controllo come una pipeline, cioè una catena di montaggio a più stadi: questa innovazione ha il grosso vantaggio di ridurre il critical path interno alla CPU e consente ai RISC di raggiungere frequenze di clock più alte rispetto agli analoghi CISC.

Nel caso di context-switch o di chiamata a subroutine o comunque di uso dello stack i RISC invece di accedere alla memoria di sistema usano un meccanismo chiamato register renaming, che consiste nel rinominare i registri in modo da usare per la nuova esecuzione una diversa zona del file di registri, senza dover accedere alla memoria ogni volta.

La complessità nei RISC si sposta dall'hardware al software: un compilatore che genera codice per CPU RISC deve affrontare un duro lavoro per generare codice compatto ed efficiente, che in ogni caso sarà più grande ed occuperà più memoria dell'equivalente per CISC.

16. Discutere brevemente le ragioni dell'introduzione delle architetture RISC e le principali differenze tra queste e le architetture tradizionali di tipo CISC.

Le ragioni dell'introduzione del RISC sono da ricercare nel fatto che le istruzioni di Load/Store sono preponderanti, quindi vanno ridotte (L/S 50%, Branch/Compare 20%, Aritmetiche 20%...). L'architettura esterna è caratterizzata da poche istruzioni e pochi modi di indirizzamento; l'accesso alla memoria è limitato alle istruzioni L/S mentre le operazioni logiche vengono svolte con operandi presenti nei registri interni. L'architettura interna è organizzata a pipeline con un'unità di controllo semplificata e l'esecuzione delle istruzioni risulta molto veloce (una per ciclo); le istruzioni hanno inoltre lunghezza fissa. CISC è l'acronimo di Complex Instruction Set Computer: tipicamente un processore di questo tipo implementa un numero relativamente scarso di registri di uso generale, ed ha una unità di controllo microprogrammata. Il set di istruzioni associato a CPU di tipo CISC è molto esteso e composto in genere di alcune centinaia di codici operativi diversi che svolgono funzioni anche molto complesse, fra cui sono caratteristici i trasferimenti memoria-memoria, assenti nei RISC; le istruzioni hanno lunghezza variabile e possono presentarsi in formati diversi, e sono necessari due o più cicli di clock per completare una istruzione. Il ridotto numero di registri interni obbliga questi processori a scrivere in memoria ogni volta che si verifica una chiamata di funzione o che viene salvato un registro nello stack.

17. Memoria virtuale

La memoria virtuale è una architettura di sistema capace di simulare uno spazio di memoria centrale maggiore di quello fisicamente presente; questo risultato si raggiunge utilizzando spazio di memoria secondaria su altri dispositivi, di solito le unità a disco. La memoria centrale fisicamente presente diventa quindi la parte effettivamente utilizzata di quella virtuale, più grande: questo stratagemma è utile in virtù del principio di località dell'esecuzione dei programmi.

In un sistema dotato di memoria virtuale, il processore e i programmi si riferiscono alla memoria centrale con indirizzi logici, virtuali, che vengono tradotti in indirizzi fisici reali da una unità apposita, la MMU o memory management unit che in genere è incorporata nei processori.

La MMU svolge i seguenti compiti: 1) Traduce l'indirizzo logico in indirizzo fisico; 2) Controlla che l'indirizzo fisico corrisponda a una zona di memoria fisicamente presente nella memoria centrale; 3) Se invece la zona in questione è nello spazio di swap, la MMU solleva una eccezione di page fault e il processore si occupa di caricarla in memoria centrale, scartando una pagina già presente.

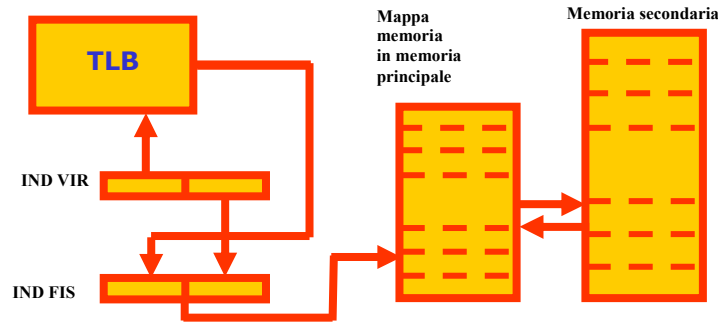
Questo meccanismo ha un prezzo in termini di prestazioni: la MMU impiega del tempo per tradurre l'indirizzo logico in indirizzo fisico, e ce ne vuole molto di più per caricare una zona di memoria dallo spazio di swap: in ultima analisi quindi, implementare una memoria virtuale significa sacrificare potenza di calcolo per poter eseguire un maggior numero di processi contemporanei.

Detto T_a il normale tempo di accesso alla memoria fisica, T_t il tempo di traduzione di indirizzi della MMU e T_{load} il tempo necessario a caricare una zona di memoria dallo swap, il tempo (medio) di accesso in caso di memoria virtuale è: $T_{av} = T_t + T_a + T_{load} * P_{fault}$

dove P_{fault} è la probabilità di page fault, cioè di incappare in una pagina che non è presente in memoria centrale e di doverla quindi caricare dallo swap.

Esistono principalmente due modi di implementare un sistema di memoria virtuale: dividere la memoria in tante pagine identiche, gestite dall'hardware, oppure lasciare che sia il programmatore e/o il compilatore che usa a "segmentare" il proprio programma in più segmenti (sperabilmente) indipendenti.

Entrambi i metodi presentano vantaggi e svantaggi: in questi ultimi tempi tuttavia il sistema di gran lunga più usato è la memoria paginata, per via della maggiore omogeneità e indipendenza dal software.



Memoria Paginata:

Con questo schema la memoria viene divisa in pagine tutte della stessa grandezza (4 o 8 kilobyte): i programmi non hanno bisogno di sapere nulla su come è organizzata la memoria e non devono avere nessuna struttura interna particolare; la esatta ubicazione e disposizione fisica della memoria che occupano non li riguarda e tutto il sistema di memoria virtuale è completamente gestito dalla MMU attraverso un complesso sistema di registri associativi. Proprio questo sistema di registri è il punto debole di questo tipo di meccanismo: se il numero delle pagine è molto grande (pagine di piccole dimensioni, oppure grandi quantità di memoria virtuale da emulare) il meccanismo associativo può diventare troppo complesso, rallentando sensibilmente l'accesso alla memoria (e quindi tutto il sistema).

Un possibile rimedio è quello di aumentare la dimensione della pagine di memoria, al prezzo di un maggior spreco di memoria stessa (frammentazione interna: più grandi sono le pagine, più aumenta il numero di pagine parzialmente vuote e più spazio viene sprecato).

Memoria Segmentata:

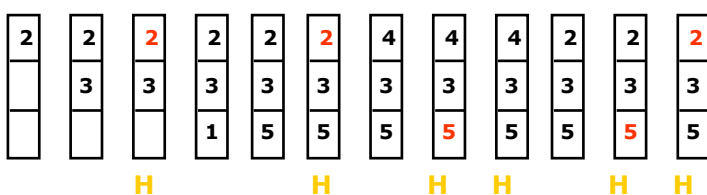
In questo caso il meccanismo di memoria virtuale è in parte software: i programmi che girano su sistemi con memoria segmentata sono strutturati in segmenti funzionalmente omogenei: la MMU tiene traccia di quali e quanti segmenti sono presenti in memoria e dove. Il vantaggio principale di questo sistema è che sfrutta al massimo il già citato principio di località, riducendo al minimo il ricorso allo spazio di swap: una volta che un programma ha in memoria centrale i segmenti di cui necessita, solo raramente ne chiederà di nuovi. Il grosso svantaggio di questo sistema, invece, è il grande spreco di memoria dovuto alla frammentazione esterna: con l'andare del tempo e il susseguirsi dei processi in esecuzione, la memoria viene allocata e disallocata in blocchi di varie dimensioni che lasciano un sempre maggior numero di "buchi" vuoti, troppo piccoli per poter essere utilmente allocati: questo rende necessario eseguire una dispendiosa compattazione periodica della memoria fisica allocata, e/o l'uso di algoritmi di allocazione molto sofisticati.

Esistono varie tecniche per decidere quali sono le aree di memoria che è preferibile spostare dalla memoria primaria alla secondaria. Le seguenti sono le più diffuse:

Strategia ottimale

Questa tecnica consiste nel rimpiazzare la pagina di memoria che verrà riutilizzata più in là nel tempo. Chiaramente, per poter essere realmente implementata, richiederebbe che il S.O. conoscesse in anticipo le pagine utilizzate nel futuro dai processi. Non è quindi utilizzabile come algoritmo di rimpiazzamento delle pagine in memoria principale, ma come metro di confronto delle altre strategie.

t	1	2	3	4	5	6	7	8	9	10	11	12
P	2	3	2	1	5	2	4	5	3	2	5	2

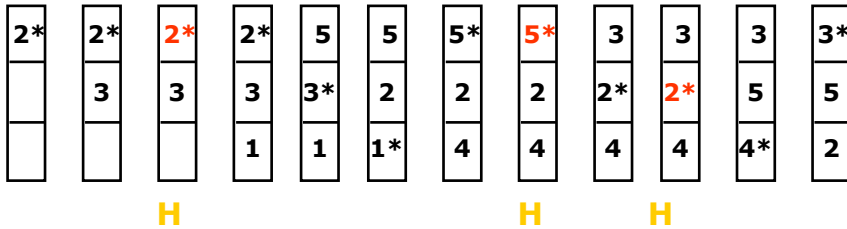


FIFO

La tecnica FIFO (First In First Out) è la più semplice, si tiene traccia in una tabella di quando è stata allocata un'area di memoria. Quando vi è una nuova richiesta di allocazione di pagine di memoria, se c'è ancora spazio in memoria principale, semplicemente viene allocata la nuova pagina, altrimenti si consulta mediante la tabella quali sono le pagine allocate da più tempo e si spostano in memoria secondaria.

Questo algoritmo è molto semplice e di rapida esecuzione ma ha lo svantaggio di spostare in memoria secondaria le pagine più vecchie anche se sono utilizzate di frequente.

t 1 2 3 4 5 6 7 8 9 10 11 12
P 2 3 2 1 5 2 4 5 3 2 5 2



LRU

La migliore soluzione possibile consisterebbe nello spostare le pagine che non saranno usate per più tempo ma naturalmente il sistema operativo non è in grado di avere quest'informazione. La soluzione di compromesso consiste nello spostare le pagine inutilizzate da più tempo (LRU cioè Least Recently Used) poiché hanno buona probabilità di non essere nuovamente utilizzate nell'immediato.

Per gestire efficientemente quest'algoritmo occorre supporto hardware. È possibile implementare questa tecnica in due modi: si può aggiornare ad ogni accesso della memoria una tabella oppure si mantiene uno stack con le pagine utilizzate più recentemente poste in cima. Entrambi i metodi hanno un impatto non indifferente sulle prestazioni del sistema e per questo motivo normalmente sono realizzati in hardware.

t 1 2 3 4 5 6 7 8 9 10 11 12
P 2 3 2 1 5 2 4 5 3 2 5 2

