Si illustrino, in generale, i concetti di località spaziale e temporale nell'esecuzione dei programmi.

I concetti di località spaziale e temporale sono strettamente correlati alla gestione della memoria e all'ottimizzazione delle prestazioni.

La località spaziale si riferisce alla tendenza di un programma ad accedere a posizioni di memoria vicine tra loro.
Ad esempio, un programma potrebbe accedere a un array utilizzando un ciclo for, accedendo agli elementi adiacenti uno dopo l'altro.

La località temporale si riferisce al fatto che un programma tende a utilizzare più volte le stesse risorse in un breve lasso di tempo.
Ad esempio, una variabile potrebbe essere letta e modificata più volte all'interno di un loop.

Sfruttare la località spaziale e temporale può ridurre il numero di accessi alla memoria principale, che sono generalmente più lenti rispetto alle cache o ai registri, contribuendo a migliorare le prestazioni complessive del programma.

Cosa succede nel caso in cui il processore voglia accedere ad una parola di memoria già presente nella cache? E cosa succede nel caso in cui non sia presente?

Quando il processore desidera accedere a una parola di memoria che è già presente nella cache, si verifica un hit di cache. In questo caso, il processore può accedere direttamente alla cache per recuperare la parola di memoria richiesta (senza passare per la memoria principale).
Dal momento che la cache è più veloce della memoria principale, l'accesso alla parola di memoria avviene in modo più rapido rispetto all'accesso diretto alla memoria principale.

Nel caso in cui la parola richiesta non sia presente nella cache, si verifica un miss di cache.
In questa situazione, il processore deve recuperare la parola di memoria dalla memoria principale. Ciò comporta un maggior ritardo in quanto l'accesso alla memoria principale è più lento rispetto all'accesso alla cache. Quando viene rilevato un miss di cache, il sistema cercherà di gestirlo riportando i dati necessari dalla memoria principale alla cache per i futuri accessi:

- l'intero processore va in stallo, bloccando il contenuto dei registri temporanei, per il tempo necessario a caricare il dato nella memoria principale;
- viene inviato il valore del PC in memoria, si richiede alla memoria principale la lettura e si attende che termini;
- si scrive il dato nel giusto blocco di cache (si aggiorna il tag e si imposta il bit di validità a 1);
- riparte l'esecuzione dell'istruzione a partire dalla fase di fetch.

In generale, l'obiettivo della gestione della cache è massimizzare l'utilizzo della cache per ridurre il numero di accessi alla memoria principale, sfruttando la località spaziale e temporale dei dati.

Quali parametri si devono considerare per valutare le prestazioni delle memorie cache?

Ci sono diversi parametri importanti:

- **Hit rate**: rappresenta la percentuale di accessi alla cache che vengono soddisfatti da un hit di cache. Un alto hit rate indica una cache efficiente e una buona località spaziale e temporale dei dati.
- **Miss rate**: rappresenta la percentuale di accessi alla cache che generano un miss di cache. Un basso miss rate indica una cache efficiente con un'alta capacità di memorizzazione dei dati frequentemente utilizzati.
- **Latenza di hit**: è il tempo richiesto per accedere a una parola di memoria presente nella cache. Poiché la cache è più veloce della memoria principale, la latenza di hit dovrebbe essere inferiore rispetto all'accesso diretto alla memoria principale.
- **Latenza di miss**: è il tempo richiesto per recuperare una parola di memoria mancante dalla cache. La latenza di miss è generalmente più alta rispetto alla latenza di hit.
- **Dimensione della cache**: Indica la quantità di dati che possono essere memorizzati nella cache. Una cache più grande tende ad avere un tasso di colpi più alto, ma richiede più spazio e può aumentare la latenza di miss.
- **Ciclo di Bus**: diminuirlo significa effettuare più trasferimenti per unità di tempo.

In quali modi avviene la mappatura della memoria cache?

La mappatura della memoria cache può avvenire in tre modi diversi: mappatura diretta, completamente associativa e set-associativa.

Per la **mappatura diretta**, a ogni blocco della cache è assegnato un indirizzo univoco di M bit (con M < N bit di indirizzo della memoria principale. A ogni blocco è assegnato un bit di validità e alcuni bit di tag.
Quando un dato presente nella memoria principale deve essere inserito nella cache, esso verrà copiato nel blocco il cui indirizzo coincide con gli ultimi M bit del suo indirizzo in memoria centrale, i restanti N-M bit vengono inseriti nel campo tag, e viene posto il bit di validità a 1.
Quando il dato dovrà essere letto, il processore avrà a disposizione l'indirizzo in memoria principale, dal quale leggerà le prime M cifre che comporranno l'indirizzo del blocco nella cache, poi controllerà il bit di validità di quel blocco, e infine confronterà i N-M bit di indirizzo rimanenti con il tag del blocco, e se coincidono avviene una hit di cache, se non coincidono, o se il bit di validità vale 0, avviene una miss di cache.

Se la cache è **completamente associativa**, ogni blocco della memoria principale può essere mappato in qualsiasi linea disponibile della cache, la ricerca avviene quindi in parallelo su ogni blocco, basandosi su bit di validità e tag.

La memoria cache **set-associativa** è un ibrido tra le due citate prima. La cache è divisa in linee, ognuna delle quali contiene diversi blocchi.
Ogni blocco di memoria principale può essere mappato in un insieme di linee della cache, anziché in una posizione specifica. Quando un blocco di memoria principale viene cercato nella cache, viene cercato all'interno del set corrispondente: con l'indirizzo si accede a una determinata linea,

come nella mappatura diretta, e all'interno della linea si ricerca il blocco desiderato tramite tag e bit di validità, come nella cache completamente associativa.
La mappatura set-associativa offre una maggiore flessibilità rispetto alla mappatura diretta, riducendo il numero di conflitti di cache.

Come avviene la scrittura dati in memoria tramite la cache?

Vi sono due modi di implementare la scrittura tramite cache: il write-through (scrittura immediata) e il write-back (scrittura posticipata).

Nel caso del write-through, ogni scrittura viene eseguita simultaneamente sia nella cache che nella memoria principale: quando un dato viene scritto nella cache, viene scritto immediatamente anche nella posizione corrispondente nella memoria principale.
Ciò comporta che la cache sia sempre coerente con la memoria principale, ma comporta un aumento nei tempi di scrittura.

Nel caso del write-back, le scritture vengono effettuate solo nella cache. Quando un blocco viene sostituito da un altro in cache, i dati vengono scritti nella memoria principale solo se sono stati modificati. La scrittura in memoria principale viene quindi posticipata fino a quando non è strettamente necessaria, riducendo il numero delle operazioni di scrittura e migliorando le prestazioni. Ovviamente in questo caso il rischio di incoerenza è elevato. Per mantenere la coerenza dei dati, i blocchi di cache modificati (dirty) devono essere marcati come tali, in modo che le scritture nella memoria principale siano effettuate solo per i blocchi necessari.

Cosa si intende con master e slave nell'ambito dei bus?

Il master è responsabile di iniziare e controllare l'operazione di trasferimento dei dati. È in grado di generare segnali di controllo per avviare la comunicazione e stabilire la sincronizzazione con gli altri dispositivi.

Lo slave risponde alle richieste di lettura o scrittura provenienti dal dispositivo attivo. Non ha il controllo diretto sul bus o sul flusso di dati, ma deve essere in grado di riconoscere le richieste di accesso alla memoria o ai registri, elaborare i dati richiesti e inviarli al dispositivo attivo.

Quando il dispositivo attivo avvia una richiesta di lettura o scrittura, invia i segnali di controllo appropriati sul bus per indicare l'operazione richiesta. Il dispositivo passivo che riceve la richiesta elabora la richiesta e invia i dati richiesti o esegue l'operazione di scrittura sul bus. Durante questo processo, i dispositivi possono scambiarsi segnali di handshaking (coordinamento e sincronizzazione) per confermare l'avvio e il completamento del trasferimento dei dati.

Come interagiscono un dispositivo attivo e uno passivo in una comunicazione tramite bus asincrono?

In una comunicazione tramite bus asincrono, i dispositivi possono essere classificati come attivi (master) o passivi (slave) in base al loro ruolo nel trasferimento dei dati.

Il dispositivo attivo inizia e controlla l'operazione di trasferimento dei dati, mentre il dispositivo passivo risponde alle richieste e fornisce i dati richiesti. La comunicazione avviene seguendo un protocollo di bus specifico che definisce i segnali di controllo e le modalità di sincronizzazione tra i dispositivi attivi e passivi.

Perché l'arbitraggio di un bus è necessario?

L'arbitraggio di un bus è necessario quando più dispositivi condividono un bus comune e desiderano accedere ad esso contemporaneamente. L'arbitraggio del bus risolve i conflitti di accesso al bus determinando quale dispositivo ha il diritto di utilizzarlo in un determinato momento.

Si spieghi cos'è l'arbitraggio di un bus con meccanismo daisy chain.

Il Daisy chain è una tecnica utilizzata per determinare l'ordine di accesso al bus tra i dispositivi connessi in cascata o a catena (i dispositivi sono collegati uno dopo l'altro, formando una linea sequenziale).

Il meccanismo daisy chain si basa sul concetto di segnali di *request* e segnali di *grant* (concessione) che vengono trasmessi lungo la catena dei dispositivi. Il segnale di request indica che un dispositivo desidera accedere al bus, mentre il segnale di grant viene utilizzato per notificare al dispositivo successivo nella catena che può accedere al bus.

Nel meccanismo daisy chain, solo un dispositivo alla volta ha l'accesso al bus.
Gli altri dispositivi nella catena devono attendere il loro turno.

Questa tecnica è particolarmente adatta quando i dispositivi hanno priorità equivalenti e l'ordine di accesso al bus non è critico.

Un esempio comune di utilizzo del meccanismo daisy chain è nel caso di dispositivi di I/O collegati in cascata.

In riferimento alle caratteristiche del livello ISA, descrivere cosa si intende per "modalità di indirizzamento".

La "modalità di indirizzamento" si riferisce alla forma in cui vengono specificati gli operandi o gli indirizzi delle istruzioni di un processore.
Le diverse modalità di indirizzamento possono influire sulla flessibilità, l'efficienza e la complessità delle istruzioni di un ISA.
La scelta delle modalità di indirizzamento dipende dalla progettazione dell'ISA stesso e dalle esigenze specifiche dell'architettura del processore.

Qual è la differenza tra ordinamento little endian e big endian dei byte all'interno di una parola?

La distinzione tra little endian e big endian è importante perché influisce su come i processori e i dispositivi interpretano e rappresentano i dati.

Nell'ordinamento little endian, i byte meno significativi (cioè quelli che rappresentano i valori numerici più bassi) sono memorizzati prima dei byte più significativi.

Nell'ordinamento big endian, invece, i byte più significativi vengono memorizzati prima dei byte meno significativi.

La scelta tra ordinamento little endian e big endian è determinata dall'architettura del processore e dai requisiti del sistema.

Nell'architettura RISC-V, quali sono i tipi di registri? Perché ci sono diversi tipi?

Nell'architettura RISC-V, ci sono diversi tipi di registri che svolgono funzioni specifiche nel processore.

- Registro zero (x0), contenente il valore 0 e non modificabile.
- Registri temporanei – *t* : utilizzati per archiviare dati e operandi durante l'esecuzione dei programmi e non vengono salvati in caso di chiamata a procedura. Sono numerati da x5-x7 e x28-x31.
- Registri da salvare – *a* : vanno da x8-x9 e x18-x27. A differenza dei temporanei, il loro contenuto va preservato se utilizzati dalla procedura chiamata.
- Registri utilizzari per il passaggio di parametri (a): sono x10-x17 e sono utilizzati per il passaggio di parametri nelle procedure e per immagazzinare il valore di ritorno.
- Registri per i risultati – *ra* (Return Address Register): registro speciale x1 che contiene l'indirizzo della prossima istruzione da eseguire (PC). Viene utilizzato per il flusso di controllo delle istruzioni, indicando quale istruzione verrà eseguita successivamente.
- Registri per i puntatori: il registro *sp* (Stack Pointer) per il puntatore dello stack, il registro *gp* (Global Pointer) per l'accesso a dati globali, il registro *fp* (Frame Pointer) per puntare alla base dello stack frame e il registro *tp* (Thread Pointer).

La divisione dei registri in diversi tipi aiuta nel design dell'architettura e nella flessibilità del processore. Separare i registri di scopo generale dai registri specializzati può semplificare l'implementazione hardware e consentire di allocare risorse specifiche per funzioni critiche come la gestione delle eccezioni o il flusso di controllo.
L'utilizzo di registri specifici consente di ottimizzare l'esecuzione delle istruzioni, semplificare la gestione del programma e migliorare le prestazioni complessive del processore.

Cosa è lo stack

È un'area di memoria che cresce e si riduce in modo dinamico in modo da consentire l'inserimento e la rimozione efficiente dei dati. La struttura dello stack segue il principio LIFO (Last-In, First-Out).
Lo stack viene utilizzato principalmente per due scopi:

- Gestione delle chiamate di procedure (quando una procedura viene chiamata, i parametri e l'indirizzo di ritorno vengono inseriti nello stack);
- Gestione delle variabili locali (quando la funzione viene chiamata, lo stack viene utilizzato per creare uno spazio per le variabili locali. Quando la funzione termina, lo spazio riservato per le variabili locali viene liberato).

L'accesso allo stack avviene tramite due operazioni principali: "push" (inserimento) e "pop" (rimozione).

Cos'è lo Stack frame?

Quando una funzione viene chiamata, uno stack frame (frame dello stack) viene creato nello stack per archiviare i dati specifici della funzione, come i parametri, le variabili locali e l'indirizzo di ritorno. Il frame pointer è un registro che punta alla base di questo frame nello stack.

Cos'è un driver?

Un driver è un software che consente la comunicazione e la gestione di un dispositivo hardware da parte del sistema operativo o di altre applicazioni.
I driver fungono da intermediari tra il sistema operativo e i dispositivi hardware, consentendo al sistema operativo di interagire correttamente con il dispositivo e utilizzare le sue funzionalità. Ogni dispositivo hardware ha il suo driver specifico, che comprende le istruzioni e le routine necessarie per inizializzare, controllare e comunicare con il dispositivo.
I driver svolgono le funzioni di:

- Inizializzazione e configurazione del dispositivo;
- Comunicazione (gestisce la trasmissione e la ricezione di dati attraverso il dispositivo hardware e ne gestisce il flusso);
- Controllo e gestione del dispositivo;
- Interfaccia con il sistema operativo.

Descrivere l'I/O basato su DMA.

Il DMA è impostato direttamente dal software o dal sistema operativo inizializzando opportuni registri.
La CPU e il DMA si contendono l'uso del bus.
L'I/O basato su DMA (Direct Memory Access) è una tecnica che consente al dispositivo di I/O di trasferire direttamente i dati tra la periferica e la memoria principale senza coinvolgere il processore principale. Invece di utilizzare il processore per gestire il trasferimento di dati, il DMA consente al dispositivo di I/O di assumere il controllo diretto sulla memoria e trasferire i dati autonomamente.
Il DMA è utilizzato per migliorare le prestazioni. È particolarmente utile per trasferimenti di grandi quantità di dati.

Descrivere l'I/O basato su busy waiting.

L'I/O basato su busy waiting (attesa attiva) è una tecnica di gestione dell'I/O in cui il processore controlla lo stato del dispositivo di I/O e lo stato della periferica in modo iterativo ispezionando in un ciclo il bit del registro di stato del controllore (READY) fino a che questi non segnala di essere pronto per un nuovo comando di I/O (l'operazione di I/O viene completata).
Durante l'attesa, il processore rimane impegnato nell'attività di verifica dello stato del dispositivo senza svolgere altre attività significative (polling).
Quando il dispositivo di I/O segnala che l'operazione è stata completata, il processore può procedere con le operazioni successive.

Descrivere l'I/O basato su interrupt.

La tecnica di Interrupt permette di liberarsi del busy waiting: il dispositivo, quando ha finito, genera un segnale (interrupt) che avverte la CPU di aver completato il proprio lavoro.

Quando un dispositivo di I/O richiede l'attenzione del processore, viene generato un interrupt che interrompe il flusso normale dell'esecuzione del programma e passa il controllo al gestore di interrupt appropriato.

Offre diversi vantaggi rispetto al polling e al busy waiting, tra cui la possibilità per il processore di essere interrotto da più dispositivi contemporaneamente.

La CPU può abilitare l'interrupt mettendo a 1 un opportuno bit.
Il problema di questa tecnica è che serve un interrupt per ogni carattere letto o scritto.

Gli interrupt possono essere generati in diversi contesti:

- Interrupt hardware: generati da dispositivi di I/O o da eventi hardware, come una pressione di un tasto sulla tastiera o un errore di hardware.

- Interrupt software: generati da richieste di servizi o eventi software, come un'eccezione di divisione per zero, una richiesta di sistema operativo o una richiesta di un'applicazione in esecuzione.

Gli interrupt sono asincroni rispetto al programma, nel senso che interrompono il flusso di esecuzione normale del programma in modo imprevedibile e non sincronizzato con le istruzioni eseguite dal processore.

Gli interrupt devono essere gestiti in modo trasparente: lo stato dell'esecuzione dopo la gestione dell'interrupt deve tornare esattamente come era prima dell'interrupt stesso.

Se ci sono più dispositivi che lavorano su interrupt, in genere si definisce un criterio di priorità tra questi dispositivi per evitare che un interrupt ne interrompa un altro. Se la CPU non permette più livelli di priorità si utilizza normalmente un chip apposito che gestisce in modo "autonomo" le interruzioni.

Cosa è il Program Counter?

Il Program Counter (PC) è un registro speciale che contiene l'indirizzo della prossima istruzione da eseguire nel programma. Il ciclo di aggiornamento del Program Counter avviene durante l'esecuzione di un programma e segue un flusso sequenziale per determinare l'indirizzo dell'istruzione successiva da eseguire.

Si descriva il ciclo di aggiornamento del Program Counter.

All'avvio del programma o quando viene raggiunta una condizione di branch o jump, il Program Counter viene inizializzato con l'indirizzo dell'istruzione di partenza del programma. Questo indirizzo è solitamente noto come indirizzo di base.

1) *Fetch*: il Program Counter ottiene dalla memoria l'indirizzo dell'istruzione successiva da eseguire e il valore del PC viene incrementato per puntare all'indirizzo successivo.
2) *Execute*: l'istruzione recuperata durante il fetch viene eseguita dal processore.
3) Aggiornamento del Program Counter: dopo l'esecuzione dell'istruzione corrente, il PC viene aggiornato per puntare all'indirizzo dell'istruzione successiva. Il modo in cui viene aggiornato dipende dal tipo di istruzione eseguita.

Se l'istruzione è di tipo sequenziale (non condizionale), il Program Counter viene semplicemente incrementato di un valore fisso (4 nel RISC-V) per puntare all'istruzione successiva.

Se l'istruzione è di tipo condizionale o di salto, il PC viene aggiornato in base alla condizione o all'indirizzo specificato dall'istruzione stessa.

Il ciclo di aggiornamento del PC si ripete iterativamente per l'esecuzione di ogni istruzione nel programma. Il fetch, l'esecuzione e l'aggiornamento del Program Counter continuano fino a

quando non viene raggiunta una condizione di terminazione del programma, ad esempio un'istruzione di fine programma o un'eccezione.

In riferimento alla traduzione di programmi, perché non basta una sola lettura del codice sorgente (passata) da parte di un assemblatore?

Il processo di traduzione del codice sorgente in codice macchina richiede più passaggi per garantire che tutte le informazioni siano correttamente risolte.
Nel codice sorgente, le etichette vengono utilizzate per identificare posizioni di memoria o punti di riferimento nel programma. Durante la traduzione, l'assemblatore deve risolvere queste etichette in indirizzi di memoria effettivi. Tuttavia, alcune etichette potrebbero essere definite in sezioni del codice che vengono utilizzate in punti successivi del programma. Pertanto, è necessario un passaggio aggiuntivo per garantire che tutte le etichette siano correttamente risolte.

A cosa serve la tabella degli opcode ad un assemblatore?

Serve a fornire un punto di riferimento per l'assemblatore affinché possa comprendere e tradurre correttamente il codice sorgente scritto in linguaggio assembly.
La tabella degli opcode contiene informazioni sui diversi tipi di istruzioni supportate dall'architettura del processore:

- codice di operazione (opcode) che identifica univocamente l'istruzione;
- formato dell'istruzione - indica la struttura e l'organizzazione dei bit all'interno dell'istruzione stessa;
- lunghezza in byte dell'istruzione nel linguaggio macchina;
- tipi di operandi supportati dall'istruzione;
- effetti che l'istruzione ha sull'architettura del processore.

Durante la fase di traduzione, l'assemblatore esamina il codice sorgente assembly linea per linea e consulta la tabella degli opcode per individuare il corrispondente opcode e le informazioni associate a ciascuna istruzione.

Cosa sono forward e backward references?

Sono concetti utilizzati per gestire la risoluzione delle etichette o dei simboli nel codice sorgente.

Una forward reference si verifica quando una particolare etichetta o simbolo viene utilizzato prima della sua definizione nel codice sorgente.
Per risolvere le forward references, è necessario un secondo passaggio dell'assemblatore per individuare le posizioni delle etichette mancanti.

Una backward reference si verifica quando un'etichetta o un simbolo viene definito prima di essere utilizzato nel codice sorgente. In questo caso, l'etichetta viene definita in precedenza e successivamente viene fatta riferimento ad essa in istruzioni o espressioni successive.

Descrivere linker, compiler e loader.

Il compilatore è un programma che traduce il codice sorgente scritto in un linguaggio di programmazione ad alto livello in codice assembler.
Dopodichè l'assemblatore potrà effettuare la traduzione da codice assembler a linguaggio macchina.

Il linker è un programma che si occupa di combinare diversi file oggetto (generati da compilatore e assemblatore) in un singolo file eseguibile.
Il linker esegue tre passi:

1) inserisce in memoria in modo simbolico il codice e i moduli dati;
2) determina gli indirizzi dei dati e delle etichette che compaiono nelle istruzioni;
3) corregge i riferimenti interni ed esterni.

Il loader si occupa di caricare il codice eseguibile in memoria e prepararlo per l'esecuzione.

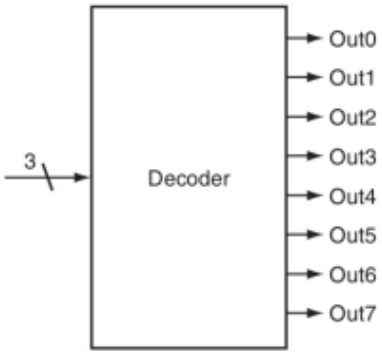
Che cosa è la gerarchia delle memorie? Parla di organizzazione delle memorie e delle diverse tecnologie.

La gerarchia delle memorie è un concetto fondamentale nell'architettura dei calcolatori che si riferisce all'organizzazione e alla disposizione delle diverse memorie all'interno di un sistema. La gerarchia delle memorie è progettata per ottimizzare l'accesso ai dati in base alla velocità, alla capacità e al costo delle diverse memorie disponibili.

le memorie sono organizzate in più livelli, da quelli più vicini al processore e più veloci, a quelli più lontani e più lenti. Di solito, la gerarchia delle memorie comprende i seguenti livelli:

- Il livello più vicino e più veloce della gerarchia delle memorie è costituito dai **registri** interni al processore.
- l livello successivo è costituito dalla **cache**, che è una memoria di dimensioni relativamente piccole, ma molto veloce, che si trova vicino al processore.
- Poi c'è la **memoria principale** (RAM). Ha una capacità maggiore rispetto alla cache, ma è più lenta in termini di velocità di accesso.
- il livello più lontano e più lento della gerarchia delle memorie è rappresentato da dispositivi di archiviazione a lungo termine, come dischi rigidi e SSD. Questi dispositivi offrono una capacità di archiviazione molto maggiore rispetto alla memoria principale, ma sono significativamente più lenti nell'accesso ai dati.

Descrivere un decoder.



a. A 3-bit decoder

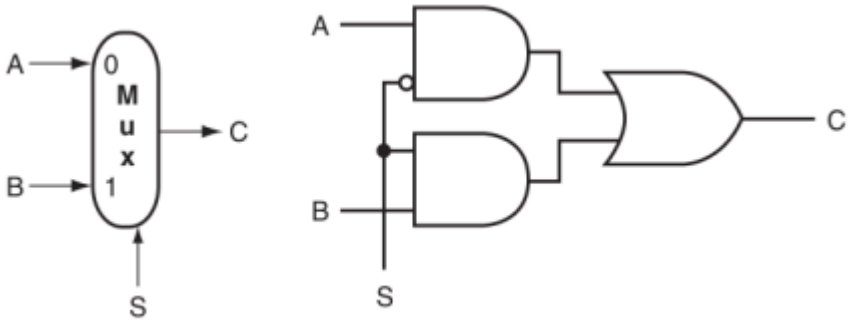
Il decoder (o decodificatore) è un blocco logico che presenta n bit in ingresso e 2n bit in uscita, dei quali solo uno sarà asserito per ogni determinata combinazione in ingresso.

Il decoder traduce gli n bit in ingresso nella corrispondente rappresentazione binaria.

Inputs			Outputs							
12	11	10	Out7	Out6	Out5	Out4	Out3	Out2	Out1	Out0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

b. The truth table for a 3-bit decoder

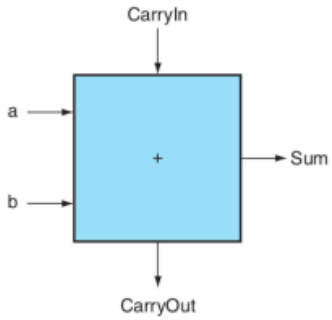
Descrivere un multiplexer.

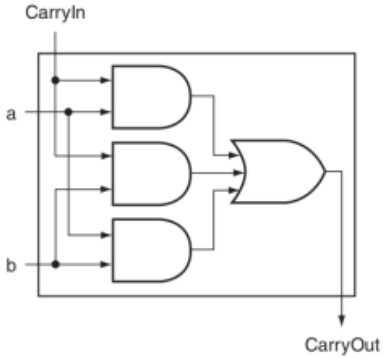


Un multiplexer (MUX) è un utilizzato per selezionare uno tra diversi segnali di ingresso e indirizzarlo verso una singola linea di uscita in base a un segnale di controllo.
I multiplexer possono gestire un numero arbitrario di linee dati in ingresso.
Se ci sono n linee dati in ingresso sono necessari $\log_2(n)$ bit di controllo. In questo caso, il multiplexer consiste essenzialmente di tre parti:

- Un decoder che genera n segnali, ognuno per indicare un differente valore in ingresso
- Un vettore di n porte AND, ognuna che combina una linea di ingresso con una uscita del decoder
- Una singola porta OR che colleziona le uscite delle porte AND

Cosa è un sommatore?





Un sommatore (adder) ha due ingressi per gli operandi, una uscita a singolo bit per il risultato e una seconda uscita a un bit per il riporto (Carry Out). È presente un ulteriore ingresso (Carry In), che prende in input il CarryOut del sommatore vicino.

Descrivere la differenza tra latch e flip flop.

I flip-flop e i latch sono gli elementi di memoria più semplici; in entrambi il segnale di uscita corrisponde al valore contenuto nell'elemento in sè. Inoltre, a differenza dei latch S-R, questi elementi sono connessi al clock di sistema, il che significa che un eventuale cambio nel loro stato è subordinato al segnale di clock.

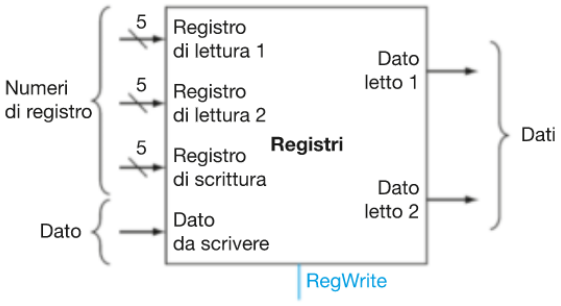
La differenza tra un flip-flop e un latch è proprio il punto del clock in cui essi aggiornano il proprio stato: in un latch il valore memorizzato cambia quando l'input cambia e il segnale di clock è asserito, mentre un flip-flop aggiorna il proprio stato solamente sul fronte del clock.

Descrivere il funzionamento del register file.

I registri universali del processore sono raccolti nel register file, un insieme di registri in cui ciascuno di essi può essere letto o scritto specificando il numero ad esso associato all'interno dell'insieme. Il register file contiene lo stato dei registri del calcolatore.

Per scrivere un dato di una word serviranno due ingressi:

- il primo deve specificare il numero del registro di scrittura;
- il secondo deve fornire il dato da scrivere.



Il register file fornisce in qualsiasi momento in uscita il contenuto del registro letto. La scrittura, invece, viene controllata da un segnale di controllo esplicito, "RegWrite". Quindi serviranno complessivamente quattro ingressi e due uscite.