Schema moltiplicazione numeri reali standard IEEE 754.

Se operandi sono 0 risultato è 0. Passo successivo sommare gli esponenti; se essi sono memorizzati in forma polarizzata, la loro somma raddoppierebbe la polarizzazione. (Valore polarizzazione deve essere sottratto dalla somma). Il risultato potrebbe comportare un underflow o un overflow dell'esponente, che verrebbe riportato provocando la conclusione dell'algoritmo. Se l'esponente del prodotto è compreso nel proprio intervallo, è necessario moltiplicare i significandi tenendo conto dei loro segni. La moltiplicazione viene eseguita come per gli interi. Concluso il calcolo del prodotto, il risultato viene normalizzato e arrotondato, così come avviene per la somma e la sottrazione. (La normalizzazione potrebbe comportare un overflow dell'esponente).

Pipeline, tecnica predizione salto utilizzando 2 bit di predizione.

Si associa ad ogni istruzione di salto condizionato uno o più bit che ne riflettano la storia recente. Tali bit, detti "taken/not taken switch", spingono il processore a prendere una particolare decisione al salto successivo. Se si utilizzano 2 bit di predizione è possibile memorizzare il risultato delle ultime 2 istanze, oppure memorizzare uno stato in qualche altro modo. L'algoritmo inizia leggendo la prossima istruzione di salto condizionato. Fino a quando le successive istruzioni di salto condizionato vengono effettuate, il processo decisionale prevede che il prossimo salto verrà intrapreso. Solo se due salti successivi non vengono intrapresi l'algoritmo deve prevedere che i salti non vengano effettuati fino a quando non siano intrapresi due salti successivi.

Architettura RISC tratta efficientemente chiamata annidata procedure.

Chiamate di procedura tipicamente coinvolgono pochi parametri e non presentano un grado di annidamento elevata, si è pensato di usare molti gruppi di registri, detti finestre di registri, per gestire le chiamate annidate. Una chiamata seleziona automaticamente un nuovo gruppo di registri e quando essa è conclusa ed effettua il ritorno, riseleziona il gruppo di registri riferito alla chiamata che l'aveva chiamata. Ogni gruppo di registri è suddiviso in tre sottogruppi: parametri passati alla procedura, registri memorizzano contenuto delle variabili locali procedura e registri temporanei che gestiscono il ritorno della procedura. Registri temporanei di un gruppo si sovrappongono perfettamente con quelli che contengono parametri del gruppo successivo, cioè del gruppo riferito ad una chiamata annidata. Tali registri sono fisicamente gli stessi e ciò permette il passaggio dei parametri senza trasferimento dei dati. La realizzazione fisica di finestre di registri sovrapposte avviene tramite buffer circolare. Quando avviene una chiamata il puntatore alla finestra corrente (CWP) viene aggiornato per farlo puntare a finestra attiva. Se si esaurisce capacità buffer (Tutte finestre sono in uso a causa di chiamate annidate) la finestra che per prima è stata inserita nel buffer viene salvata in memoria principale e sovrascritta dalla nuova. Quando una procedura termina, una finestra viene liberata e grazie ad un apposito puntatore (SWP) è possibile ripristinare l'ultima finestra salvata in memoria principale.

Divisione numeri reali standard IEEE754.

Verificare la presenza di 0. Se il divisore è 0, errore, o risultato posto a infinito. Un dividendo pari a 0 produce un risultato nullo. Esponente del divisore viene sottratto da esponente del dividendo. Ciò rimuove la polarizzazione, che deve essere risommata. Vengono poi eseguiti dei controlli sull'underflow o l'overflow dell'esponente. Il passo successivo consiste nel dividere i significandi e poi si esegue la normalizzazione e l'arrotondamento.

Hardware adottato realizzare somma e sottrazione numeri interi rappresentati in complemento a due.

L'elemento centrale dell'hardware è il sommatore binario (adder) al quale vengono forniti gli addendi o sottraendi e minuendi. Esso può produrre in risultato della somma o sottrazione oppure un overflow, che genera un flag che viene inserito in un apposito registro. Per la somma i due numeri provengono da due registri ed il risultato può essere memorizzato in uno dei due o in un terzo. Nella sottrazione al sottraendo viene applicato il complemento a due e il sommatore legge questa nuova modifica.

Codifica complemento a due numeri interi. Problemi legati realizzazione moltiplicazione di due interi rappresentati in complemento a 2, esemplificando tali problemi su un caso concreto di moltiplicazione.

Attraverso rappresentazione in complemento a due, con n bit a disposizione possiamo rappresentare tutti i numeri interi da -2^(n-1) a +2^(n-1)-1. Segno numero viene indicato mediante il bit più a sinistra, il quale rappresenta – se è a 1 e + se è a 0. Rappresentazione per i numeri positivi è identica a quella in modulo e segno. Per rappresentare un numero negativo vengono utilizzati due metodi: si calcola la sua versione positiva in binario e si complementa a 2 (complemento a 1 e poi somma di 1); si calcola la sua versione positiva in binario, si scrivono gli stessi bit da destra a sinistra fino al primo 1, e i numeri a sinistra di tale 1 vengono complementari. Nella moltiplicazione di due interi rappresentati in complemento a due nascono problemi legati al bit più significativo. Se nella moltiplicazione di due numeri è presente almeno un numero negativo, il suo bit di segno verrà calcolato nella moltiplicazione ed andrà a rendere errato il risultato. Per risolvere tali problemi bisogna utilizzare la rappresentazione in complemento a due per i prodotti parziali. La soluzione adottata è l'algoritmo di Booth.

Implementazione istruzioni attraverso tecnica della microprogrammazione. Dire se questa tecnica viene utilizzata per i processori CISC o RISC, motivare risposta.

Ad ogni codice operativo si fa corrispondere indirizzo di inizio di un µprogramma. Nell'implementazione µprogrammata fase di fetch e fase di execute possono essere descritte in un linguaggio di µprogrammazione. Ad ogni istruzione macchina viene associato un µprogramma che è formato da una sequenza di µistruzioni. Esse sono formate da micro-ordini (ognuno corrispondente ad un segnale di controllo) registrati nella memoria ROM (read only memory) detta anche memoria di controllo ed organizzati in una word chiamata word di controllo. Nella word di controllo ogni bit corrisponde ad un micro-ordine, ovvero rappresenta una linea di controllo. Le unità di controllo µprogrammate si possono suddividere in due categorie:

- µprogrammazione orizzontale quando le microistruzioni sono composte da un numero elevato di bit e quindi possono essere svolti molti compiti in parallelo, generando svariati segnali di controllo contemporaneamente;
- µprogrammazione verticale quando le microistruzioni presentano un numero limitato di bit. La microprogrammazione verticale conduce ad una minore velocità di funzionamento, in quanto, diminuendo il numero di bit, si ha bisogno di più microistruzioni per specificare tutte le operazioni svolte con una sola microistruzione "orizzontale".

Nell'unità di controllo µprogrammata il codice operativo dell'istruzione in linguaggio macchina indirizza una locazione della memoria di mapping nella quale è registrato l'indirizzo di partenza del corrispondente µprogramma. Il µPC contiene l'indirizzo della memoria di controllo e il µIR contiene la microistruzione.

La microprogrammazione è utilizzata per implementare l'Unità di Controllo della CPU la quale, grazie al microprogramma, implementa ogni istruzione tramite una sequenza di micro-operazioni eseguite direttamente dall'hardware e di generare nella giusta sequenza i segnali di controllo che provocano l'esecuzione di ogni operazione elementare. Il microprogramma (firmware) nell'unità di controllo ha una struttura ciclica in cui alterna l'esecuzione di una operazione speciale con l'esecuzione di un'operazione esterna il cui codice e dati da elaborare sono stati acquisiti dalla operazione speciale. Il microprogramma riunisce quindi tutte le micro-operazioni necessarie per effettuare l'operazione speciale del microprogramma e le micro-operazioni necessarie per effettuare ogni operazione esterna. In generale la parte operativa invia all'unità di controllo delle variabili di condizionamento. In base a queste variabili, l'Unità di controllo manda dei segnali di controllo (α β) che designano la micro-operazione da eseguire. La microprogrammazione è la soluzione tipica di architetture CISC per implementare l'Unità di Controllo. Questo perché l'adozione di tale tecnica permette una maggiore flessibilità nella progettazione, cioè rende facile modificare le sequenze di micro-operazioni che implementano le istruzioni eseguite dalla CPU, e premette la realizzazione di un vasto numero di istruzioni da parte della CPU.

Pipeline MIPS, in che modo lo stadio ID è in grado di rilevare dipendenza dati.

Quando istruzione passa da fase ID a quella EX si dice che l'istruzione è stata "rilasciata" (issued). È possibile individuare tutte le dipendenze dai dati nella fase ID. Se si rileva una dipendenza dai dati per una istruzione, questa va in stallo prima di essere rilasciata. È possibile determinare che tipo di data forwarding adottare per

evitare lo stallo. Una osservazione chiave è che i registri di pipeline contengono: dati su cui effettuare il forwarding: campi registro sorgente e destinazione. Tutti i dati su cui effettuare il forwarding provengono: output ALU e memoria dati, e sono diretti verso: input ALU, memoria dati e il comparatore con 0. Quindi occorre confrontare i registri destinazione di IR in EX/MEM e MEM/WB con i registri sorgente di IR in ID/EX e EX/MEM.

Rappresentazione numeri reali standard IEEE754.

Rappresentazione in virgola mobile è definita nello standard IEEE 754, sviluppato per facilitare portabilità programmi e per incoraggiare sviluppo di programmi sofisticati dal punto di vista numerico. Ampliamente usato in tutti i processori moderni. Definisce formato singolo a 32 e doppio a 64 bit, rispettivamente con esponenti da 8 e 11 bit. La base è 2, e l'1 è implicito a sinistra della virgola. I formati estesi includono bit aggiuntivi nell'esponente e nel significando, usati per i calcoli intermedi, grazie alla loro precisione superiore diminuiscono la possibilità di un risultato finale contaminato da eccessivi errori di arrotondamento. Una motivazione aggiuntiva per il formato esteso è che racchiude alcuni benefici del formato doppio senza incorrere nella penalità di tempo associata solitamente a una precisione più alta. Non tutte le combinazioni di bit nei formati IEEE sono interpretate nel solito modo; infatti alcune sono usate per rappresentare le seguenti classi di numeri:

- Esponenti nell'intervallo da 1 a 254 in formato singolo e da 1 a 2046 in formato doppio.
- Esponente 0 con una frazione di 0 rappresenta lo zero positivo o negativo.
- Esponente di tutti 1 con una frazione di 0 rappresenta l'infinito positivo o negativo.
- Esponente 0 con frazione non nulla rappresenta un numero denormalizzato (in questo caso il bit a sinistra è 0 e l'esponente è -126 0 -1022).
- Esponente di tutti 1 con una frazione non nulla ha il valore simbolico NaN (not a number).

Pipeline, descrivere problematica dipendenza dati e discutere tecniche per trattare il problema.

I tipi di conflitto generabili sono 3:

- (RAW, read after write): quando istruzione modifica registro o locazione di memoria e istruzione successiva legge dato in quella locazione di memoria o quel registro.
- (WAR, write after read): istruzione legge registro o locazione di memoria e istruzione successiva scrive nella stessa posizione.
- (WAW, write after write): due istruzioni devono scrivere nella stessa locazione. Seconda istruzione deve essere ritardata di tanti cicli di clock quanti sono richiesti per rimuoverne la dipendenza.

Le possibili soluzioni possono essere:

- Introduzioni di fasi non operative (dette nop).
- Individuazione rischio e prelievo del dato direttamente dall'uscita dell'ALU (data forwarding).
- Risoluzione a livello del compilatore (tramite architettura MIPS).
- Riordino delle istruzioni (pipeline scheduling).

Confronta modo in cui architettura RISC usa ampio banco registri rispetto alla gestione di una cache.

Banco registri organizzato in finestre agisce come un piccolo buffer che conserva alcune delle variabili che hanno un'alta probabilità di essere usate con maggiore frequenza. Banco registri basato su finestre contiene le variabili scalari locali delle N-1 procedure più recentemente attivate. Cache contiene una selezione delle variabili scalari usate di recente. Il banco dei registri è più veloce, ma la cache usa lo spazio in modo più efficiente, dato che opera dinamicamente. Inoltre generalmente le cache trattano tutti i riferimenti alla memoria allo stesso modo, incluse le istruzioni e altri tipi di dato. Il banco dei registri fa un uso inefficiente dello spazio quando non tutte le procedure richiedono l'intera dimensione della finestra ad esse allocata. La cache è in grado di trattare variabili locali e globali. Con il banco dei registri il trasferimento dati tra registri e memoria è determinato dalla profondità di annidamento della procedura. Poiché tale profondità solitamente varia poco, l'uso della memoria è relativamente poco frequente.

Approcci per trattare indirizzo ritorno chiamata di procedura.

Una procedura è un programma autonomo incorporato all'interno di un programma più grande. Può essere invocata in ciascun punto del programma. Processore viene istruito per andare a eseguire l'intera procedura e poi ritorna al punto in cui si è verificata la chiamata. Ragioni principali per l'uso di procedure sono risparmio e modularità. Essa prevede due istruzioni di base: una chiamata che provoca il salto dalla locazione attuale della procedura, e una di ritorno che rimanda dalla procedura al punto della chiamata.

- Primo approccio del registro: esempio: CALL X provoca:□RN←PC + D (D=lunghezza istruzione); PC←X in cui la procedura chiamata può salvare i contenuti di RN per usarli per il successivo ritorno.
- Il secondo approccio consiste nel memorizzare indirizzo di ritorno all'inizio della procedura. In questo caso: CALL X provoca: X←PC + D ; PC←X+1□e l'indirizzo di ritorno è stato memorizzato in modo sicuro.
- Ultimo approccio consiste nell'usare una pila. Quando il processore esegue una chiamata, posiziona gli indirizzi di ritorno in cima alla pila e vengono presi nell'ordine inverso alla chiusura delle procedure.

L'indirizzo di ritorno da una chiamata di procedura permette all'istruzione di Return della procedura di tornare nell'appropriata sezione di codice, cioè in quella dove era avvenuto il Call. Tale indirizzo può essere memorizzato in un registro, all'inizio della procedura chiamata o in cima alla pila. La memorizzazione nel registro prevede l'utilizzo di un registro specializzato dove verrà salvato l'indirizzo dell'istruzione successiva a quella che ha fatto la chiamata. Verrà poi caricato nel PC l'indirizzo della prima istruzione della procedura chiamata, che al return andrà a leggere il registro specializzato. Detti: RN = registro specializzato, Δ = lunghezza di una istruzione, PC = program counter, X = indirizzo della procedura chiamata. Volendo schematizzare: RN = PC + Δ PC = X La memorizzazione all'inizio della procedura chiamata salva l'indirizzo dell'istruzione successiva a quella che ha chiamato la proceduta all'inizio della procedura chiamata. Viene poi copiato sul PC l'indirizzo della seconda istruzione della procedura, contenendo la prima l'indirizzo di ritorno e la seconda la vera istruzione da eseguire. Volendo schematizzare: X = PC + Δ PC = X + 1Questi due metodi di memorizzazione non permettono la gestione di chiamate annidate, la quale può avvenire solo mediante la memorizzazione in cima alla pila.Con la memorizzazione in cima alla pila gli indirizzi di ritorno vengono memorizzati uno dopo l'altro in cima alla pila, e vengono presi nell'ordine inverso all'inserimento alla chiusura delle procedure. La pila è una porzione di memoria riservata dove le scritture e le letture avvengono sempre in cima. Il top della pila è indicizzato da un apposito registro della CPU, lo stack pointer SP. Questo tipo di memorizzazione permette la gestione di chiamate annidate. Ad esempio avendo il Call della procedura 1 l'indirizzo di ritorno di 1 viene registrato sul top della pila, ad un Call annidato di una procedura 2 l'indirizzo di ritorno di 2 viene registrato sul top della pila ed l'indirizzo di ritorno di 1 va in seconda posizione. Quando 2 farà il return sulla pila leggerà il corretto indirizzo, cancellandolo, a questo seguirà il return di 1 la quale troverà ancora il corretto indirizzo, cancellandolo.

Architettura CISC (Complex Instruction Set Computer).

In seguito all'evoluzione dei calcolatori si riscontrò un costo del software molto maggiore rispetto al costo dell'hardware. I linguaggi di programmazione ad alto livello stavano diventato sempre più potenti e complessi. Questo fenomeno originò un altro problema, il gap semantico, differenza tra le operazioni consentite dagli HLL e quelle fornite dall'architettura del calcolatore. Come conseguenza ne derivarono inefficienza dell'esecuzione, dimensione eccessiva del codice e complessità dei compilatori. Gli scopi dell'architettura CISC consistevano nel:

- Facilitare scrittura del compilatore.
- Migliorare efficienza esecuzione.
- Supportare linguaggi ad alto livello più complessi.

Tramite CISC si ottengono programmi più piccoli e più veloci a causa un utilizzo sempre minore di memoria, di un numero sempre minore di istruzioni (e quindi di codice da prelevare), e di un utilizzo minore di pagine, riducendo la probabilità di un page fault.

In che modo compilatore aiuta utilizzo efficace dei registri con architettura RISC.

Ipotizzando che solo un piccolo numero di registri sia disponibile su una data macchina RISC, l'uso ottimale dei registri è lasciato al compilatore. Programmi scritti ad alto livello non hanno espliciti riferimenti a registri, ma

fanno riferimento a variabili in maniera simbolica. Ogni quantità che nel programma è candidata a risiedere in registro è assegnata a un registro simbolico. Compilatore mappa un numero virtualmente illimitato di registri simbolici su registri reali del processore. Registri simbolici il cui uso non si sovrappone temporalmente possono condividere (essere mappati) stesso registro reale. Se registri non sono sufficienti per contenere tutte variabili riferite in un dato intervallo di tempo, alcune variabili vengono mantenute in memoria principale. La tecnica comunemente più usata per decidere quali quantità debbano essere assegnate ai registri in ogni dato punto del programma è la "colorazione di un grafo". Dato un grafo, costituito da nodi connessi di archi, si assegnano dei colori ai suoi nodi in modo che nodi adiacenti non abbiano lo stesso colore, e che il numero dei colori usati sia minimo. I nodi del grafo corrispondono a registri simbolici. Due registri che sono in "vita" all'interno di uno stesso frammento di codice sono connessi da un arco. L'idea di fondo è colorare il grafo con n colori, dove n è il numero di registri reali. I nodi che non possono essere colorati sono perciò memorizzati in memoria principale.

Dipendenze controllo di una pipeline.

La dipendenza da controllo è un problema che avviene quando si ha a che fare con un'istruzione che altera in qualche modo la normale sequenzialità delle istruzioni. Queste istruzioni quali salti condizionati e non, chiamate o ritorni a procedure o interruzioni, modificano il contenuto del program counter e quindi la fase di fetch dell'istruzione successiva a quella corrente può caricare un'istruzione sbagliata. Per trattare questo problema sono stati considerati vari approcci:

- Flussi multipli: consiste nella replicazione delle parti della pipeline in modo che una contenga l'istruzione successiva a quella corrente (nel caso che il salto non avvenga) e l'altra l'istruzione destinazione del salto chiamata target. In questo modo si ha la possibilità di caricare le istruzioni da eseguire in entrambi i casi. Una alla fine viene eseguita completamente e l'altra rimossa. Vantaggi: pipeline sempre a regime. Conveniente se i salti sono pochi. Svantaggi: costo della duplicazione delle parti. Restano comunque conflitti all'accesso di risorse (registri/memoria). Inefficace e troppo costoso per più salti condizionali in sequenza.
- Prelievo anticipato della destinazione: Quando si incontra un salto condizionato si può anticipare il fetch del target, indipendentemente che sia preso oppure no, in modo che dopo la valutazione della condizione l'istruzione a cui saltare è già presente nella CPU. Però la pipeline continua a caricare le istruzioni successive a quelle di salto, che vengono rimosse nel caso che il salto sia preso. Quest'accorgimento riduce i tempi di attesa, anche se può comportare sprechi.
- Predizione del salto: si cerca di prevedere durante il tempo di esecuzione se il salto sarà intrapreso oppure no.
- Salto ritardato: utilizza il tempo per calcolare l'indirizzo a cui saltare per fare qualcosa di utile.
- Buffer circolare: Si utilizza una memoria piccola e molto veloce, che è il buffer circolare, dove contenere le ultime n istruzioni prelevate. In caso di salto, si controlla se l'istruzione destinazione è già presente nel buffer, così da evitarne il fetch. Questa tecnica è particolarmente efficace per le istruzioni che saltano in avanti di poche istruzioni, poiché sarebbe maggiore la probabilità di trovarle nel buffer.

Pipeline descrivere problematica dipendenza controllo e in particolare tecnica buffer circolare.

Dipendenza da controllo si verifica quando la pipeline intraprende decisione errata su una predizione di salto e di conseguenza porta nella pipeline istruzioni che devono essere successivamente eliminate. Tutte le istruzioni che perciò modificano il PC (salti condizionati e non, chiamate a e ritorni da procedure, interruzioni) invalidano la pipeline. La fase fetch successiva carica l'istruzione seguente che non può essere quella giusta. Le soluzioni possono essere: mettere in stallo la pipeline fino a quando non si è calcolato l'indirizzo della prossima istruzione (pessima efficienza ma massima semplicità), individuare le istruzioni critiche per anticiparne l'esecuzione, eventualmente mediante apposita logica di controllo. Le soluzione del buffer circolare per i salti condizionati consiste nell'utilizzare una memoria piccola e molto veloce dove mantenere le ultime n istruzioni prelevate. In caso di salto, si controlla se l'istruzione destinazione è già presente nel buffer, così da evitare il fetch della stesa. I vantaggi:

- Anticipando il fetch, alcune delle istruzioni successive a quella corrente saranno già presenti nel buffer e se non si ha salto non ci sarà bisogno di caricarle in memoria.

- Se si salta in avanti di poche istruzioni, l'istruzione destinazione sarà già presente nel buffer.
- Se il salto condizionale realizza un ciclo le cui istruzioni possono essere tutte contenute nel buffer, non c'è bisogno di effettuare fetch ripetuti delle sue stesse istruzioni.

Descrivere formati codifica di istruzione, specificando composizione tipica, pregi, difetti.

Uno dei progetti più semplici per un calcolatore ad uso generale era il PDP-8, il quale adottava istruzioni a 12 bit e operava su parole a 12 bit. Per i dati temporanei c'era solo un registro detto accumulatore. Ciascun accesso alla memoria consisteva di 7 bit più 2 modificatori da 1 bit. La memoria era divisa in pagine di 128 parole ciascuna. Il calcolo dell'indirizzo si basava su riferimenti alla pagina 0 e alla pagina corrente. Le istruzioni del PDP-8 sono notevolmente efficienti in quanto supporta l'indirizzamento indiretto, quello con spiazzamento e l'indicizzazione. In contrasto con questo progetto è nato il PDP-10, ossia un sistema a condivisione di tempo. Esso era basato su i principi di ortogonalità, completezza ed indirizzamento diretto. Esso prevedeva parole e istruzioni di 36 bit. Il codice operativo occupava 9 bit, consentendo fino a 512 operazioni. L'indirizzo che specifica uno dei 16 registri occupa 4 bit. L'altro riferimento all' operando inizia con un campo indirizzo in memoria da 18 bit. Le sue istruzioni facilitano il compito del programmatore e del compilatore a spese di un utilizzo inefficiente dello spazio. Il PDP-11 è stato progettato per un linguaggio macchina potente e flessibile che rispettasse i requisiti di un microcomputer a 16 bit. Esso dispone di 8 registri generici da 16 bit (uno viene usato come puntatore alla pila). Inoltre gode della indipendenza definita ortogonalità. Lo svantaggio è che il costo hardware e la complessità della programmazione sono piuttosto alti. Il formato delle istruzioni VAX invece è stato progettato adottando due criteri: 1) tutte le istruzioni dovrebbero avere un numero "naturale" di operandi, 2) tutti gli operandi dovrebbero presentare la stessa generalità nelle specifiche. Il risultato consiste in un codice operativo di 1 o 2 byte seguito da alcuni specificatori di operando a seconda del codice operativo. Esso gode quindi di un'ampia gamma di operazioni e di modi di indirizzamento, il conto però da pagare è un aumento nella complessità del processore rispetto a uno con un formato e un insieme di istruzioni più semplici.

Discutere nel dettaglio in cosa consiste il formato variabile per le istruzioni. Dare esempi di formati variabili.

Definisce disposizione dei bit, i termini delle sue parti costituenti, un codice operativo e, in modo implicito o esplicito, zero o più operandi. Il **PDP-11** è stato progettato per un linguaggio macchina potente e flessibile che rispettasse i requisiti di un microcomputer a 16 bit. Esso dispone di 8 registri generici da 16 bit (uno usato come puntatore pila). Gode della indipendenza definita ortogonalità. I formati comprendono istruzioni a zero, uno e due indirizzi. Lunghezza codice operativo varia dai 4 ai 16 bit. I riferimenti ai registri sono lunghi 6 bit. Istruzioni del PDP-11 sono lunghe una parola (16 bit), oppure da 32 e 48 bit. Il formato delle istruzioni **VAX** invece è stato progettato adottando due criteri:

- Tutte le istruzioni dovrebbero avere un numero "naturale" di operandi
- Tutti gli operandi dovrebbero presentare la stessa generalità nelle specifiche.

Il risultato consiste in un codice operativo di 1 o 2 byte seguito da alcuni specificatori di operando a seconda del codice operativo. Lunghezza minima di istruzione è 1 byte (è sufficiente per descrivere la maggior parte delle istruzioni), ma si possono costruire istruzioni da 37 byte. Il numero di specificatori di operando può arrivare a 6. Uno di essi occupa 1 byte in cui i 4 più a sinistra specificano il modo di indirizzamento. Uno specificatore di operando spesso consiste di un solo byte, ove i 4 più a destra specificano uno dei 16 registri generici.

Spiegare la differenza tra la codifica dei numeri interi in com. a 2 rispetto a quella in modulo e segno.

Bit più significativo come bit di segno (1 per meno). La forma più semplice di rappresentazione che adotta un bit di segno è detta in modulo e segno. In una parola da n bit, gli n-1 bit più a destra contengono il modulo del numero. Gli svantaggi sono però che per eseguire correttamente somme e sottrazioni occorre considerare i segni dei due numeri e i loro moduli.Può essere considerata come somma pesata di bit. Sia A un numero di n bit in complemento a due. Se A è positivo allora il bit di segno corrisponde a 0 (da 0 (n zeri) a 2^{n-1} -1), se A è negativo invece il bit di segno è 1 (da -1 a -2^{n-1}).

Si descrivano i formati delle istruzioni MIPS visti a lezione, discuterne caratteristiche, pregi, e difetti.

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

La MIPS prevede 3 formati di istruzione:

- Registro (R), utilizzato per le operazioni logico-aritmetiche.
- Load/store (I),per le istruzioni load e store che scambiano i dati tra la memoria e i registri. Può essere utilizzato anche per le operazioni logico-aritmetiche dove un operando è immediato (ad esempio una ADDI) e in tal caso un operando va in rs e il dato immediato nell'address.
- Jump (J), per le istruzioni di salto.

Il formato R ha i campi: op, rs, rt, rd, shamt, funct. Il campo rd contiene il registro di destinazione, dove viene messo il risultato dell'operazione, mentre gli operandi vengono messi nel rs ed rt. Op rappresenta il codice operativo, shamt la quantità di shift e funct il tipo di operazione. Il formato I ha i campi: op, rs, rt, address.Rt contiene il contenuto del registro che deve essere caricato dalla memoria o in memoria, rs contiene il contenuto del registro base, address il dato immediato. Il formato J ha i campi: op, address dove address contiene l'indirizzo di salto.

Spiegare cos'è il salto ritardato.

E' una soluzione che permette di utilizzare il tempo per calcolare l'indirizzo a cui saltare per fare qualcosa di utile. Di fatto il salto non viene ritardato ma l'intervallo precedente a esso viene usato per eseguire una qualche istruzione che sia indipendente dal salto stesso. Il compilatore decide quale, dopo aver analizzato il programma, e la pone nella locazione di memoria branch delay slot.

Si descriva in dettaglio le modalità di indirizzamento indiretto. Discuterne pregi e difetti. La adozione di tale modo di indirizzamento è favorito o sfavorito in un'architettura RISC? Motivare la risposta.

Nell'indirizzamento indiretto ogni istruzione è suddivisa nei campi op-code ed indirizzo. Il campo indirizzo contiene l'indirizzo di una cella di memoria, la quale contiene l'indirizzo dell' operando. Il vantaggio di tale tecnica è la possibilità di indirizzare un grande quantità di celle, precisamente 2k con k la lunghezza del campo indirizzo. Lo svantaggio di tale metodo è che esso richiede due accessi in memoria per ottenere l' operando. L'adozione di tale modo di indirizzamento è sfavorito in un'architettura RISC, in quanto tale architettura ottimizza l'accesso alle variabili locali mediante l'utilizzo di un ampio numero di registri e utilizza un set di istruzioni semplificato, cercando di minimizzare gli accessi in memoria.

Si descrivano nel dettaglio le modalità di indirizzamento con spiazzamento e a pila. In particolare si confrontino criticamente i due modi di indirizzamento e se ne discutano pregi e difetti.

Lo spiazzamento è la combinazione di indirizzamento diretto con indirizzamento registro indiretto.Il campo indirizzo ha due sottocampi A=valore di base,R=registro che contiene l'indirizzo di un valore da sommare ad A per ottenere l'indirizzo.

Pipeline, descrivere tecnica data-forwarding: a cosa serve? Come funziona? Che supporto hw ha bisogno?

Data-forwarding è una **soluzioni** utilizzate per **superare dipendenze** dei dati che si possono verificare tra istruzioni in una pipeline. Se viene individuato una dipendenza dai dati e il tipo di dipendenza lo permette, il data forwarding, permette di trasferire i dati dall'output della ALU in ingresso alla ALU. Questo è possibile grazie ad appositi **circuiti di ByPass e MUX**, regolati da Unità di Controllo e da altre unità, che permettono alla ALU di caricare dati derivanti dalla memoria o dati che la ALU stessa ha mandato in output nel ciclo precedente.

Nella realtà il data-forwarding può essere implementato in diversi modi che dipendono dall'architettura.Esempio pratico lo troviamo nell'architettura MIPS in cui vi è un apposito circuito di identificazione delle dipendenze e gestione del data-forwarding. Tale circuito ha tre componenti fondamentali: **Forwarding Unit**: unità che decide se attivare il forward attivando nell'opportuno modo i multiplexer della ALU Hazard, **detection Unit**: unità in grado di riconoscere le dipendenze e di generare stalli in caso di dipendenze non risolvibili, **Control Unit**: manda segnali di controllo che regolano il forward. Nel caso in cui la dipendenza venga rilevata come risolvibile, allora nelle MIPS sarà possibile fare il forward di dati da fase EX a EX e da fase MEM a EX.

Formato delle istruzioni

Formato istruzioni descrive campi istruzione, lunghezza e numero indirizzi. In qualsiasi formato è incluso il codice operativo, che decide quale operazione fare, e 0, 1 o più operandi in modo implicito o esplicito. Formato istruzioni può essere a lunghezza: **fissa,** tutte le istruzioni hanno la stessa lunghezza, ma posso avere più formati cambiando i campi. Questo formato è estremamente efficiente nell'uso della pipeline; **variabile,** ogni istruzione ha lunghezza che dipende dal numero operandi e nel campo con l'opcode devo anche specificare il numero di operandi. Permettono una grande flessibilità, ma incrementano notevolmente la complessità; **ibrida,** ha diversi formati con lunghezza fissa che è data da un compromesso tra repertorio istruzioni potente e necessità di risparmiare spazio ed è condizionata da diversi fattori: dimensione memoria (deve poter essere completamente indirizzata), organizzazione della memoria, struttura del bus (in base a quanto è capiente posso discriminare numero di accessi), complessità CPU e velocità richiesta della CPU. Allocazione dei bit nei campi indirizzo dipende da: numero dei metodi d'indirizzamento, numero di operandi, numero di registri (più sono, più vengono utilizzati, minore sarà il numero di bit utilizzati per il più costoso indirizzamento a memoria), numero di banchi di registro, intervallo di indirizzi da rendere disponibile, granularità d'indirizzamento. Vantaggi e svantaggi: grande varietà istruzioni e lunghezze differenti permettono utilizzo vasto numero di opcode e metodi di indirizzamento, permettendo una grande flessibilità. Questo tipo di istruzioni può essere realizzato utilizzando istruzioni a lunghezza variabile, ma il loro utilizzo complica notevolmente la CPU, rende difficile il fetch e globalmente diminuisce la reattività operazione più frequentemente utilizzate. Tale caratteristiche si avvicinano alla filosofia CISC. Numero fisso formati mancanza di ortogonalità(operandi indipendenti dal codice operativo) tra opcode e metodo d'indirizzamento e numero limitato operandi utilizzabili per operazioni. Mancanza di flessibilità. CPU più semplice da realizzare, permette caricamento istruzioni in modo uniforme veloce e dimensione fissa delle istruzioni favorisce uso della pipeline. Tale caratteristiche si avvicinano alla filosofia RISC. Possibili formati codifica istruzione sono: **PDP-8:** architettura molto vecchia in cui era disponibile un solo registro, accumulatore. Nonostante le forti limitazioni vi è un indirizzamento abbastanza flessibile. Istruzioni erano a lunghezza fissa, con molti formati, che permettevano un estensione degli opcode grazie all'utilizzo di microperazioni. Un formato ottimizzato al massimo per quel tempo. **PDP-10:** un unico formato a lunghezza fissa, le stesse tipologie di operazioni per diversi tipi di operandi, solo indirizzamento diretto ed introdotto il concetto di ortogonalità. Facilitava il lavoro dei programmatori e dei compilatori ma non utilizzava in modo efficiente lo spazio a disposizione.**PDP-11:** adotta una formato ibrido in cui ogni operando può utilizzare un qualsiasi tipo di indirizzamento. Ha un'ampia variabilità di formati e metodi di indirizzamento che la rendono costosa e complicata da realizzare, ma i programmi risultano essere efficienti e compatti. **VAX:** un sistema estremamente variabile con lunghezza variabile e formati diversi, in quanto l'opcode può stare su 1 o 2 byte. E' un sistema molto flessibile e potente che facilita il lavoro di programmatori e compilatori, ma il sistema è molto complesso. **PENTIUM:** le istruzioni sono composte da vari pezzi i quali vengono assemblati in base alle necessità. Il risultato sono quindi istruzioni a lunghezza variabile e di vari formati che richiedono una complessa decodifica. Tale approccio è stato utilizzato per mantenere la retro compatibilità. **PowerPC:** un sistema con istruzioni a lunghezza fissa e diversi formati, avendo una suddivisione non omogenea dei campi. Pensato per computer RISC.

Formato variabile per istruzioni.

Formato variabile per istruzioni permette grande varietà istruzioni e grande flessibilità. Tali istruzioni hanno lunghezze variabili, un campo opcode espanso, così da permettere la codifica di un vasto numero di operazioni e metodi di indirizzamento ed inoltre nel campo opcode viene anche specificare il numero di operandi presenti

nell'istruzione. Istruzione formato variabile può avere numero variabile di campi di tipo diverso, come ad esempio nel PENTIUM. Uso di questo tipo di istruzioni complica notevolmente CPU, più costosa, rende difficile fetch e globalmente diminuisce la reattività sulle operazione più frequentemente utilizzate. Tale caratteristiche si avvicinano alla filosofia CISC. Due esempi formati variabili istruzioni: VAX: un sistema estremamente variabile con lunghezza variabile e formati diversi, in quanto l'opcode può stare su 1 o 2 byte. E' un sistema molto flessibile e potente che facilita il lavoro di programmatori e compilatori, ma il sistema è molto complesso. PENTIUM: le istruzioni sono composte da vari pezzi i quali vengono assemblati in base alle necessità. Il risultato sono quindi istruzioni a lunghezza variabile e di vari formati che richiedono una complessa decodifica. Tale approccio è stato utilizzato per mantenere la retro compatibilità.

Discutere le motivazioni alla base dei processori multicore.

I microprocessori hanno visto una crescita esponenziale delle prestazioni grazie al miglioramento dell'organizzazione ed all'incremento delle frequenza di clock. Il miglioramento dell'organizzazione del cip è stato fortemente focalizzato sull'incremento delle parallelismo tra istruzioni. Si è passati infatti dall'introduzione della pipeline, a CPU superscalari, in cui vi sono pipeline parallele, sino a CPU con multithreading simultaneo (SMT), in cui alle pipeline parallele sono associati banchi di registri replicati.Tali miglioramenti hanno però richiesto un aumento di complessità, dalla quale segue quindi una logica più complessa quindi difficile da realizzare, progettare e verificare, ed un aumento dell'area del chip per permettere di supportare il parallelismo. Inoltre all'aumentare della densità del cip è seguito un aumento esponenziale della potenza richiesta, quindi maggiore energia consumata e maggior calore prodotto. Con le CPU SMT si era giunti al limite della potenza erogabile ed ai limiti del parallelismo a livello di istruzioni, quindi per permettere un aumento delle capacità delle CPU si è scelto di passare ad architetture multicore.Si ipotizza infatti che con le architetture multicore sia possibile un incremento prestazionale quasi lineare, anche se i vantaggi prestazionali dipendono dallo sfruttamento efficace delle risorse parallele da parte dei programmi (piccole quantità di codice seriale ha un impatto significativo sulle prestazioni).

Possibili alternative organizzazione processore multicore.

Organizzazione processore multicore dipende da: numero di core per cip, tipologia di core (sei i singoli core sono superscalari o SMT)(core più vecchi avevano organizzazione superscalare, mentre le CPU multicore più recenti hanno organizzazione SMT), numero di livelli di cache per chip (che possono essere L1,L2,L3), quantità di cache condivisa divide i multicore in 4 macrogruppi:

- Cache L1 dedicata: ogni core ha la propria cache L1 dedicata, la quale è suddivisa tra cache dati e cache istruzioni
- Cache L2 dedicata: ogni core ha la propria cache L1 ed L2 dedicata.
- Cache L2 condivisa: ogni core ha la propria cache L1 dedicata, ma vi è una cache L2 condivisa tra tutti i core.
- Cache L3 condivisa: ogni core ha la propria cache L1 ed L2 dedicata, ma vi è una cache L3 condivisa tra tutti i core.

L'utilizzo di cache L2 condivisa ha i seguenti vantaggi:

- 1) Interferenza costruttiva: un processo accede alla memoria e carica dei dati. Tali dati servono ad un altro processo su un altro core, il quale troverà i dati già caricati sulla cache condivisa. Vi è quindi una riduzione accidentale del numero di miss.
- 2) Dati condivisi tra più core non sono replicati a livello di cache condivisa, ma potrebbero esserlo nella cache non condivisa.
- 3) Grazie ad opportuni algoritmi di sostituzione dei blocchi, la cache può essere dedicata dinamicamente ad ogni core. (Quindi processi con minore località possono utilizzare più cache)
- 4) Le comunicazione dentro al processore sono più facili da realizzare
- 5) Il problema della coerenza dei dati viene confinata nella cache L1.