Possibili domande Architettura. 2° parte.

1. Spiegare nel dettaglio lo schema per realizzare la moltiplicazione tra numeri reali dello standard IEEE 754.
2. Moltiplicazione: innanzitutto se gli operandi sono 0 il risultato è 0. Il passo successivo consiste nel sommare gli esponenti; se essi sono memorizzati in forma polarizzata, la loro somma raddoppierebbe la polarizzazione. Quindi il valore della 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).
3. Discutere nel dettaglio in cosa consiste il formato variabile per le istruzioni. Dare esempi di formati variabili.
4. Abbiamo visto diversi progetti di macchine storiche che hanno utilizzato diversi formati di istruzioni. Il formato di un’istruzione definisce la disposizione dei suoi bit, in termini delle sue parti costituenti e deve includere 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 viene usato come puntatore alla pila). Inoltre gode della indipendenza definita ortogonalità. I formati comprendono istruzioni a zero, uno e due indirizzi. La lunghezza del codice operativo varia dai 4 ai 16 bit. I riferimenti ai registri sono lunghi 6 bit. Le istruzioni del PDP-11 sono lunghe solitamente una parola (16 bit), oppure da 32 e 48 bit. 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. La lunghezza minima di un’istruzione è 1 byte(che è sufficiente per descrivere la maggior parte delle istruzioni del VAX), 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.
5. Nel contesto di una pipeline, descrivere nel dettaglio la tecnica della predizione di un salto utilizzando 2 bit di predizione.

3) Per prevedere se un salto verrà intrapreso possono essere utilizzate varie tecniche. E’ possibile associare a 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 alla successiva occorrenza dell’istruzione. Se si utilizzano 2 bit di predizione è possibile memorizzare il risultato delle ultime 2 istanze di esecuzione dell’istruzione associata, 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. Se una singola previsione è errata, l’algoritmo continua a prevedere che il prossimo salto verrà effettuato. 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.

1. Spiegare nel dettaglio come un’architettura RISC possa trattare efficientemente la chiamata annidata di procedure

4)

1. Spiegare in dettaglio la divisione fra i numeri reali secondo lo standard IEEE754

5) Divisione: Il primo passo è verificare la presenza di 0. Se il divisore è 0, viene inviata una segnalazione di errore, o il risultato viene posto a infinito, a seconda dell’implementazione. Un dividendo pari a 0 produce un risultato nullo. Poi, l’esponente del divisore viene sottratto dall’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.

1. Descrivere sinteticamente l’implementazione delle istruzioni attraverso la tecnica della microprogrammazione. Dire se questa tecnica viene utilizzata per i processori CISC o RISC, e motivare la risposta.

6) Implementazione microprogrammata: ad ogni codice operativo si fa corrispondere l'indirizzo di inizio di un microprogramma.

Nell'implementazione microprogrammata la fase di [fetch](http://it.wikipedia.org/wiki/Fetch) e le fasi di execute possono essere descritte in un linguaggio di microprogrammazione.

Ad ogni istruzione macchina viene associato un microprogramma che è formato da una sequenza di microistruzioni. Esse sono formate da microordini (ognuno corrispondente ad un segnale di controllo) registrati nella memoria [ROM](http://it.wikipedia.org/wiki/Read-Only_Memory) (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 microordine, ovvero rappresenta una linea di controllo. Le unità di controllo microprogrammate si possono suddividere in due categorie: unità di controllo realizzate con:

* microprogrammazione 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;
* microprogrammazione 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 microprogrammata il codice operativo (C.O.) dell'istruzione in linguaggio macchina indirizza una locazione della memoria di mapping (memoria che contiene gli indirizzi di partenza dei microprogrammi) nella quale è registrato l'indirizzo di partenza del corrispondente microprogramma. Il µPC contiene l'indirizzo della memoria di controllo e il µIR contiene la microistruzione.

1. Nel contesto della pipeline MIPS, si illustri in che modo lo stadio ID è in grado di rilevare la dipendenza dei dati.

7)Quando una istruzione passa dalla fase ID a quella EX si dice che la istruzione è stata “rilasciata” (issued). Nella Pipeline MIPS 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. Inoltre, sempre nella fase ID, è possibile determinare che tipo di data forwarding adottare per evitare lo stallo ed anche predisporre gli opportuni segnali di controllo. La logica per decidere come effettuare il forwarding è simile a quella appena vista per individuare le dipendenze, ma considera molti più casi. Una osservazione chiave è che i registri di pipeline contengono: dati su cui effettuare il forwarding. i campi registro sorgente e destinazione. Tutti i dati su cui effettuare il forwarding provengono: dall’output della ALU e dalla memoria dati, e sono diretti verso: l’input della ALU, l’input della 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.

8) Spiegare in dettaglio la rappresentazione dei numeri reali secondo lo standard IEEE754

8) La più importante rappresentazione in virgola mobile è definita nello standard IEEE 754, sviluppato per facilitare la portabilità dei programmi e per incoraggiare lo sviluppo di programmi sofisticati dal punto di vista numerico. Lo standard è ampliamente usato in tutti i processori moderni. Questo standard definisce un formato singolo a 32 bit e un formato doppio a 64, 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. Essi devono essere utilizzati per i calcoli intermedi. Grazie alla loro precisione superiore diminuiscono la possibilità di un risultato finale contaminato da eccessivi errori di arrotondamento: con il loro intervallo più ampio, essi diminuiscono anche la possibilità di un overflow intermedio il cui risultato sarebbe rappresentabile in un formato di base. 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.   
-un esponente 0 con una frazione di 0 rappresenta lo zero positivo o negativo.   
-un esponente di tutti 1 con una frazione di 0 rappresenta l’infinito positivo o negativo, a seconda del bit di segno.  
-un 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).  
-un esponente di tutti 1 con una frazione non nulla ha il valore simbolico NaN (non a number).

9) Nel contesto di una pipeline descrivere la problematica della dipendenza dei dati e si discutano in dettaglio le tecniche viste a lezione per trattare il problema.

9) Un conflitto sui dati si verifica quando esiste un conflitto nell’accesso alla locazione di un operando. I tipi di conflitto generabili sono 3: 1) Scrittura-Lettura (RAW, read after write), o dipendenza effettiva: quando un’istruzione modifica un registro o una locazione di memoria e un’istruzione successiva legge il dato in quella locazione di memoria o quel registro. 2) Lettura-Scrittura (WAR, write after read), o antidipendenza: un’istruzione legge un registro o una locazione di memoria e un’istruzione successiva scrive nella stessa posizione. 3) Scrittura-Lettura (WAW, write after write), o dipendenza di output: due istruzioni devono scrivere nella stessa locazione. Se si verifica una dipendenza di due istruzioni nel caso WAR, la seconda istruzione deve essere ritardata di tanti cicli di clock quanti sono richiesti per rimuoverne la dipendenza. (un’istruzione deve essere ritardata fino a che non sono prodotti i suoi input). In classe abbiamo visto che possibili soluzioni possono essere:   
-introduzioni di fasi non operative (dette nop),   
-individuazione del rischio e del prelievo del dato direttamente dall’uscita dell’ALU (data forwarding),   
-risoluzione a livello del compilatore (tramite architettura MIPS),   
-riordino delle istruzioni (pipeline scheduling).

10) Mettere a confronto il modo in cui un’architettura RISC utilizza l’ampio banco di registri a sua disposizione rispetto alla gestione di una cache.  
  
10) Il banco dei registri organizzato in finestre agisce come un piccolo buffer che conserva alcune delle variabili che hanno un’altà probabilità di essere usate con maggiore frequenza. Proprio per questo esso assomiglia molto ad una memoria cache. Essi hanno però diverse differenze. Il banco dei registri basato su finestre contiene le variabili scalari locali delle N-1 procedure più recentemente attivate. La 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.

11) Possibili approcci per trattare l’indirizzo di ritorno di una chiamata di procedura.

11) Una procedura è un programma autonomo incorporato all’interno di un programma più grande. In ciascun punto del programma la procedura può essere invocata, o chiamata. Il processore viene istruito per andare a eseguire l’intera procedura e poi ritorna al punto in cui si è verificata la chiamata. Le due ragioni principali per l’uso di procedure sono il risparmio e la modularità. Essa prevede due istruzioni di base: una chiamata che provoca il salto dalla locazione attuale alla procedura, e una di ritorno che rimanda dalla procedura al punto della chiamata. Esistono diversi approcci per trattare l’indirizzo di ritorno di questa. Il primo consiste nell’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 l’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. L’ultimo approccio consiste nell’usare una pila. Quando il processore esegue una chiamata, posiziona l’indirizzo di ritorno sulla cima della pila (porzione di memoria dove le letture/scritture avvengono sempre in cima). Gli indirizzi di ritorno vengono memorizzati in cima alla pila uno dopo l’altro, e vengono presi nell’ordine inverso alla chiusura delle procedure.

12) Motivazioni di base dell’architettura CISC.

12) Le architetture “Complex Instruction Set Computer” (ovvero CISC) nacquero per diversi motivi. Innanzitutto in seguito all’evoluzione dei calcolatori si riscontrò un costo del software molto maggiore rispetto al costo dell’hardware. Inoltre i linguaggui di programmazione ad alto livello stavano diventato sempre più potenti e complessi. Questo fenomeno originò un altro problema, il gap semantico, ossia la differenza tra le operazioni consentite dagli HLL e quelle fornite dall’architettura del calcolatore. Come conseguenza ne derivarono l’inefficienza dell’esecuzione, la dimensione eccessiva del codice e la complessità dei compilatori. Gli scopi dell’architettura CISC consistevano infatti nel 1) facilitare la scrittura del compilatore, 2) migliorare l’efficienza dell’esecuzione, 3) supportare i linguaggi ad alto livello più complessi. Tramite il 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.

13) Spiegare in che modo un compilatore possa aiutare l’utilizzo efficace dei registri da parte di un’architettura RISC.

13) Ipotizzando che solo un piccolo numero di registri sia disponibile su una data macchina RISC, l’uso ottimale dei registri è lasciato al compilatore. I programmi scritti ad alto livello non hanno espliciti riferimenti a registri, ma fanno riferimento alle variabili in maniera simbolica. Ogni quantità che nel programma è candidata a risiedere in un registro è assegnata a un registro simbolico (o virtuale). Quindi esso mappa (un numero virtualmente illimitato) di registri simbolici su registri reali del processore). I registri simbolici il cui uso non si sovrappone temporalmente possono condividere lo stesso registro reale, cioè possono essere mappati sullo stesso registro reale. Se i registri non sono sufficienti per contenere tutte le 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 possono essere colorati sono perciò memorizzati in memoria principale.

14) Nel contesto di una pipeline descrivere la problematica della dipendenza dal controllo e si discuta in particolare la tecnica del buffer circolare, spiegando in quali situazioni tale tecnica è particolarmente efficace.

14) La dipendenza da controllo si verifica quando la pipeline intraprende una 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 (tali istruzioni sono circa il 30% del totale medio di un programma). Le soluzioni possono essere o: mettere in stallo la pipeline fino a quando non si è calcolato l’indirizzo della prossima istruzione (pessima efficienza ma massima semplicità), o: 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 (appunto il buffer circolare) 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 sono i seguenti: 1)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  
2)se si salta in avanti di poche istruzioni, l’istruzione destinazione sarà già presente nel buffer.  
3)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.

15) Si descrivano i possibili formati di codifica di un’istruzione, specificando per ogni formato la sua composizione tipica, i pregi, i difetti.

15) 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 definità 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.

16) Spiegare la differenza tra la codifica dei numeri interi in complemento a 2 rispetto a quella in modulo e segno

16) Esistono varie convenzioni per rappresentare i numeri interi, e tutte trattano il bit più significativo come bit di segno. Se il bit di segno è 1 il numero è negativo, altrimenti è positivo o nullo. 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. Anche nella rappresentazione in complemento a due il bit più significativo indica il segno, ma essa differisce sotto altri aspetti. Per venir compresa può essere considerata come somma pesata di bit. Sia A un numero di n bit in co plemento a due. Se A è positivo allora il bit di segno corrisponde a 0 ( da 0 (n zeri) a 2n-1 -1), se A è negativo invece il bit di segno è 1 (I restanti n-1 bit possono assumere 2n-1 configurazioni diverse, quindi da -1 a -2^(n-1).

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

17) La MIPS è un esempio di architettura RISC con pipeline ottimizzata. Le istruzioni occupano 32 bit e non si fa uso di codici di condizione 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).