**Descrivere la differenza tra architettura e organizzazione di un calcolatore**

L’architettura di un elaboratore è costituita dall’insieme degli attributi visibili al programmatore:

− il repertorio delle istruzioni, il numero di bit usati per rappresentare i dati, i meccanismi di I/O, le tecniche di indirizzamento della memoria.

L’organizzazione invece riguarda gli aspetti hardware trasparenti al programmatore:

− segnali di controllo, interfacce fra periferiche e calcolatore, tecnologia della memoria

Gli attributi architetturali sono gli elementi hardware che costituiscono la struttura dell’elaboratore, quelli organizzativi sono dati dal modo in cui i primi vengono implementati.

Per esempio, la disponibilità o meno dell’operazione moltiplicazione è un aspetto architetturale, il modo in cui viene implementata (circuito dedicato o somme ripetute) organizzativo.

**Descrivere in dettaglio il ciclo di fetch/execute delle istruzioni**

IR (instruction register): registro che contiene il codice operativo dell’istruzione correntemente in esecuzione.

MBR (memory buffer register): contiene una parola che deve essere immagazzinata in memoria o è pervenuta dalla memoria.

MAR (memory address register): contiene l’indirizzo della parola di memoria in cui scrivere il contenuto di MBR o che deve essere trasferita in MBR.

IBR (instruction buffer register) : contiene temporaneamente l’istruzione destra di una parola di memoria.

PC (program counter): contiene l’indirzzo della prossima coppia di istruzioni da caricare dalal memoria.

AC (accumulator) e MQ (multuplier quotient): contengono temporaneamente gli operandi e i risultati parziali delle operazioni della ALU.

Durante il ciclo di prelievo (fetch cycle), il codice operativo della successiva istruzione viene caricato nell’IR e la porzione di indirizzo viene caricata nel MAR.

Questa istruzione può essere letta da IBR o può essere ottenuta dalla memoria caricando una parola nel MBR e poi a seguire nell’IBR, nell’IR e nel MAR.

Quando il codice operativo è stato registrato nell’IR, ha inizio il ciclo di esecuzione (execute cycle). Il circuito di controllo interpreta il codice operativo ed esegue l’istruzione, impostando i segnali di controllo appropriati per trasferire i dati o per fare eseguire un’operazione della ALU.

**Spiegare a cosa serve il bus di sistema, com’è strutturato e in che modo viene utilizzato dal calcolatore.**

Il bus di sistema è il mezzo di comunicazione che collega le principali componenti dell’elaboratore: processore, memoria e unità di i/O.

Il bus consiste di più percorsi di comunicazione, detti linee, ognuna delle quali è in grado di trasmettere segnali che rappresentano una cifra binaria (bit 0 o 1). Poiché ogni linea può trasportare solo un bit alla volta, il numero di linee determina quanti bit possono essere trasportati contemporaneamente.

Le linee possono essere di 3 tipi: **dati**, **indirizzi**, **controllo**. Il numero di linee per tipo è detto ampiezza del bus.

Le linee indirizzi vengono impiegate per designare la destinazione dei dati presenti nel bus dati.

Se il processore intende leggere un dato da una locazione di memoria, ne pone l’indirizzo sulla linea degli indirizzi.

Le linee di controllo servono a controllare l’accesso e l’uso delle linee dati e indirizzi. Trasmettono sia comandi che specificano le operazioni da compiere, sia segnali di temporizzazione che indicano la validità delle informazioni da dati e indirizzi.

**Descrivere in dettaglio la gestione da programma I/O.**

Nell’I/O da programma il processore ha il controllo totale dell’operazione di I/O: invia il comando al modulo di I/O appropriato e attende finchè l’operazione non è terminata.

Per inviare il comando al modulo il processore scrive sul bus indirizzi l’indirizzo univoco associato alla periferica interessata, che verrà interpretato dal modulo per determinare quale.

Il processore deve controllare periodicamente lo stato del modulo finchè non rileva il completamento dell’operazione.

Essendo il processore più rapido del modulo di I/O, quest’attesa comporta uno spreco di tempo.

I tipi di comandi che la CPU invia al modulo sono:

-controllo: per attivare una periferica e comunicarle cosa fare.

-test: per testare le condizioni di stato dei moduli di I/O e delle sue periferiche.

-lettura: per dire al modulo, che ha scritto i dati della periferica in un buffer, di scriverli sul bus dati.

-scrittura: per dire al modulo di prendere i dati dal bus dati e trasmetterli alla periferica.

**Descrivere in che modo vengono gestite le interruzioni (sia per la componente hardware che per quella software) nel caso di I/O interrupt driven.**

Il processore invia il comando di I/O al modulo, prosegue nell’esecuzione di altre istruzioni e al termine di ognuna controlla se il modulo ha finito.

Dopo che il modulo ha depositato i dati nei suoi registri, invia al processore un interrupt tramite il bus di controllo e attende finchè il processore gli richiede i dati. Quando riceve la richiesta, il modulo scrive i dati sul bus dati.

Quando il processore rileva un interrupt inviato dal modulo di I/O, salva il contesto esecutivo del programma corrente, elabora l’interrupt e ripristina il contesto esecutivo quando ha finito.

Quando una periferica completa un’operazione di I/O, avviene la seguente sequenza di eventi:

-Il dispositivo invia un interrupt al processore.

-Il processore controlla gli interrupt riscontrandone l’eventuale presenza e invia un segnale di riconoscimento al dispositivo che ha inviato l’interrupt, che di conseguenza può rimuoverlo.

-Il processore trasferisce il controllo al programma di gestione dell’ interrupt. Salva il contesto esecutivo del programma corrente (sulla pila di sistema) dal punto in cui è sopraggiunto l’interrupt.

-Il processore scrive nel PC l’indirizzo della prima istruzione della routine di gestione dell’interrupt considerato.

-Finita la gestione dell’interrupt, rimette il contesto al suo posto e continua il programma interrotto.

**Descrivere la gestione di I/O tramite DMA**

Il DMA (direct access memory) è un modulo hardware addizionale che consente lo scambio immediato dei dati tra il modulo di I/O e la memoria centrale senza il coinvolgimento del processore.

Quando la CPU desidera leggere o scrivere un blocco di dati, invia un comando al modulo DMA, che gli restituisce un segnale di interrupt quando ha terminato.

Prima di delegare una determinata operazione di I/O al DMA, il processore gli comunica le seguenti informazioni:

-lettura/scrittura

-indirizzo del dispositivo di I/O interessato

-indirizzo iniziale in memoria del blocco coinvolto nell’operazione

-quantità di dati da trasferire

**Descrivere in dettaglio il ciclo di esecuzione con trattamento delle interruzioni**

Prima di tutto, il trattamento delle interruzioni è quel meccanismo tramite il quale altri moduli (es I/O) possono interrompere la normale sequenza di esecuzione.

Ciò è necessario perché molti dispositivi esterni sono più lenti del processore, che deve attendere la fine del loro lavoro comportando uno spreco di tempo.

Gli interrupt sono quindi introdotti sostanzialmente per migliorare l’efficienza dell’elaborazione.

Il ciclo di esecuzione con trattamento delle interruzioni avviene in questo modo:

-La CPU controlla se ci sono interruzioni pendenti.

-Se no, preleva la prossima istruzione.

-Se sì: sospende l’esecuzione del programma corrente e salva il suo contesto; imposta il PC all’indirizzo del programma di gestione dell’interruzione; esegue il programma di gestione dell’interruzione; ripristina il contesto precedente e riprende il programma interrotto.

**Spiegare la differenza fra interruzioni multiple ed interruzioni annidate discutendo criticamente le differenti modalità di trattamento da loro richieste.**

Per trattare gli interrupt multipli si possono usare due approcci.

Il primo è disabilitare gli interrupt durante l’elaborazione di un interrupt. In questo modo la CPU gestisce la prima interruzione ignorando le successive, che rimangono pendenti e controllate solo in seguito al completamento della prima.

Questo approccio è molto semplice, infatti gli interrupt vengono gestiti sequenzialmente.

Lo svantaggio principale però è che non si tiene conto di fattori di priorità e tempistica.

E qui c’è la sostanziale differenza con le interruzioni annidate. Il secondo approccio infatti consiste nel definire le priorità per gli interrupt. In questo modo, interruzioni con bassa priorità possono essere interrotte da interruzioni con priorità più alta. Quando un interrupt a priorità più alta è stato gestito, la CPU ritorna all’interruzione precedente.

**Si descrivano le caratteristiche e le operazioni principali delle memorie a semiconduttore, considerando sia memorie RAM che ROM.**

L’elemento di base di una memoria a semiconduttore è la cella di memoria. Essa presenta due stati stabili che rappresentano il bit 0 o 1, e può essere selezionata per le operazioni di lettura o scrittura, effettuate tramite segnali elettrici.

I più comuni tipi di memoria a semiconduttore sono la RAM e la ROM.

Entrambe sono ad accesso casuale.

Una caratteristica distintiva della RAM risiede nella sua volatilità. La RAM infatti deve ricevere un’alimentazione costante altrimenti i dati vanno persi, dunque essa può essere utilizzata solo per una memorizzazione temporanea.

Un’altra caratteristica è la possibilità sia di scrittura che di lettura dati in modo semplice e rapido.

La RAM inoltre può essere statica (SRAM) o dinamica (DRAM).

Dalla parte opposta abbiamo la ROM che, a differenza della RAM, è una memoria di sola lettura e non è volatile. La memorizzazione dei dati infatti è permanente, e quindi i dati restano salvati anche in assenza di alimentazione. Solitamente le ROM sono impostate al momento della loro fabbricazione o programmabili una sola volta (PROM).

Esistono varianti delle memorie ROM di sola lettura che possono essere cancellate per intero (EPROM) o parzialmente (EEPROM) e riprogrammate. La loro cancellazione avviene elettricamente tramite esposizione a raggi ultravioletti.

**Spiegare in dettaglio le differenze tra DRAM ed SRAM discutendone vantaggi e svantaggi**

Una RAM dinamica memorizza i dati in forma di cariche su condensatori, che rappresentano la presenza o assenza di carica dal valore binario 1 o 0.

Poichè essi tendono naturalmente a scaricarsi, le DRAM richiedono periodicamente il refresh delle cariche per mantenere la memorizzazione dei dati.

La costruzione del circuito è meno complessa delle SRAM, per questo motivo sono anche meno costose ma anche più lente; inoltre necessitano dei circuiti per il refresh.

Queste memorie vengono comunemente usate per la memoria principale.

La RAM statica invece memorizza ogni bit attraverso circuiti flip-flop.

E’ anch’essa volatile ma a differenza della DRAM non necessita dei circuiti di refresh.

La sua costruzione è più complessa e costosa ma è nettamente più veloce della DRAM. Essa infatti viene usata solitamente come memoria cache.

**Descrivere in dettaglio le memorie DRAM**

La memoria DRAM è composta da celle formate da un transistor e un condensatore. Il condensatore memorizza il bit 0 o 1, mentre il transistor funge da interruttore e permette di leggere il condensatore o modificargli lo stato.

Per altre informazioni leggere sopra.

**Nel contesto di una gerarchia di memoria, spiegare i possibili modi di realizzazione del mapping dei blocchi, discutendo criticamente vantaggi e svantaggi di ogni modo.**

Indirizzamento diretto:

Assegna a ciascun blocco di memoria una sola possibile linea di cache. I 3 campi indirizzo sono tag, linea e parola. Dunque il campo linea punta a una sola linea di cache, e poi si confrontano i due tag.

Vantaggi: -Semplicità di traduzione da indirizzo di memoria a indirizzo cache

-Determinazione veloce di hit/miss

Svantaggi: -Necessità di contraddistinguere il blocco con il tag

-Swap frequenti per accesso a dati di blocco adiacenti.

Associazione completa (fully associative):

I blocchi di memoria possono essere caricati in qualsiasi linea di cache. Dunque se necessario il tag del blocco deve essere confrontato con tutti i tag delle linee.

Vantaggi: Massima efficienza di allocazione

Svantaggi: Determinazione onerosa della corrispondenza tra l’indirizzo del blocco e quello della

linea e della verifica di hit/miss.

Associazione a gruppi (N-way set associative):

Cache ed MC sono organizzate in insiemi (set), ognuno dei quali contiene N blocchi (N grado di associatività).

Il campo set individua un unico set della cache e ogni blocco può essere allocato liberamente all’interno di un determinato insieme in N modi.

Vantaggi/svantaggi:

Compromesso che unisce i punti di forza del diretto e del fully riducendo al contempo i loro svantaggi.

Buona efficienza di allocazione a fronte di una sopportabile complessità di ricerca.

**Discutere le ragioni per cui è stato sviluppato il sistema RAID e descrivere in dettaglio i vari livelli**

Il sistema RAID è stato sviluppato per disporre di una grande capacità di memoria e allo stesso tempo garantire una maggiore sicurezza per la conservazione dei dati.

Questo è reso possibile dalla realizzazione di un supporto composto da più hard disk e dalla ridondanza, permettendo il recupero di dati in caso di guasto (un guasto a uno dei dischi colpisce solo quello).

L’insieme dei dischi fisici sono visti dal sistema operativo come un unico dispositivo logico.

Esistono varie modalità di organizzazione per questi dischi, che sono chiamate livelli e vanno da 0 a 6:

RAID 0:

I suoi dischi sono divisi in strisce nelle quali i dati sono distribuiti a rotazione, con un algoritmo di round robin, che parte dal primo disco, percorre gli altri fino all’ultimo e poi riprende dal primo.

Non c’è ridondanza, quindi se si perde l’informazione contenuta in una striscia si perdono anche le altre strisce dello stesso disco, e non c’è modo di recuperarla.

I dischi eseguono la ricerca dei settori in parallelo, garantendo una lettura più efficiente rispetto a quella che sarebbe offerta da unico disco molto grande, e una maggiore velocità di accesso all’informazione.

Richieste multiple di dati hanno bassa probabilità di coinvolgere lo stesso disco, diminuendo i conflitti di risorse e aumentando la velocità di ricerca che è più frequentemente in parallelo.

RAID 1:

La ridondanza viene effettuata duplicando tutti i dati. Vengono fatte due copie di dati che vengono memorizzati su dischi separati. La lettura e scrittura avviene su entrambi i dischi.

E’ garantita una grande sicurezza dei dati in quanto un recupero in caso di guasto è molto semplice, perché basta sostituire il disco non funzionante e ricopiare l’informazione.

E’ però un sistema molto costoso poiché richiede doppie risorse.

RAID 2:

Un sistema RAID 2 è composto da dischi sincronizzati tra loro, per consentire un accesso parallelo sincronizzato su di essi (la testina di ciascun disco si trova nella stessa posizione su ogni disco) e memorizzare in maniera distribuita una quantità di informazione molto piccola (spesso singolo byte) sui vari dischi; per accedere all’informazione completa bisogna accedere simultaneamente su tutti i dispositivi.

Usa un codice di Hamming per la correzione d’errore, grazie al quale si possono ricostruire le informazioni perse.

Questo sistema non è commercializzato poichè la sincronizzazione è una complicazione sia dal punto di vista elettronico che meccanica.

RAID 3:

Simile al RAID 2 ma interviene sul problema della sincronizzazione.

Solo un disco ridondante, indipendentemente dal numero di dischi. Invece dei codici di correzione utilizza dei semplici bit di parità per ogni insieme corrispondente di bit.

Dati presenti su un disco difettoso possono essere ricostruiti a partire dai dati sui dischi rimanenti e dalle informazioni sulla parità.

Velocità di trasferimento molto alta.

RAID 4

Ogni disco opera indipendentemente.

Ottimo per alti ritmi di richieste input e output.

Abbiamo bisogno di un parity disk che contiene le informazioni di parità relativa agli altri dischi. E’ calcolata la parità dei blocchi da 0 a 3.

L’unità di memorizzazione non è di singoli byte/bit ma blocchi.

Non commercializzato perché si crea collo di bottiglia perché i dischi sono tutti indipendenti tra di loro e ogni volta che si fa una modifica su uno di essi si deve aggiornare la parità del disco di parità.

RAID 5

Simile a RAID 4 ma la parità viene distribuita su tutti i dischi.

L’allocazione dei blocchi di parità avviene con round robin su tutti i vari dischi ed evita il collo di bottiglia.

Soluzione comunemente utilizzata sui server di rete.

RAID 6

Per ragioni di sicurezza si possono avere due parità calcolate in modo diverso.

Mamorizzata in blocchi separati su dischi differenti.

Se l’utente richiede N dischi, ne occorrono N+2.

Alta affidabilità sui dati.

**Nel contesto di una gerarchia di memoria spiegare come funzionano le politiche di scrittura write-through e write-back. Per ogni politica discutere criticamente i problemi che possono sorgere nell’adottarla.**

Write through:

Ogni dato modificato nella cache viene contemporaneamente modificato nella memoria centrale. In questo modo i dati sono sempre coerenti tra i livelli di memoria.

Lo svantaggio principale è che frequenti scritture sul medesimo blocco comportano un aumento di traffico nel bus con conseguente collo di bottiglia.

Write-back:

La scrittura in memoria centrale avviene solo quando il corrispettivo blocco in cache deve essere rimpiazzato.

Ciò consente un’ottimizzazione del traffico ma causa periodi di incoerenza, inoltre occorre ricordare se sono avvenute modifiche nel blocco tramite dirty bit.

**Nel contesto di una gerarchia di memoria, spiegare come i miss possono essere categorizzati in diversi tipi e dire quali strategie, per ogni tipo, si possono adottare per diminuirne il numero.**

Nel contesto di una gerarchia di memoria, i miss possono essere categorizzati in tre tipi:

 miss di primo accesso: sono invitabili e non riducibili, in quanto ogni blocco dev'essere copiato almeno una volta da memoria centrale a cache

 miss per capacità insufficiente: avvengono in quanto la cache non può contenere tutti i blocchi necessari all'esecuzione del programma

 miss per conflitto: avvengono con associazione diretta o a gruppi in quanto più blocchi possono essere mappati in uno stesso gruppo. Tale raggruppamentopuò portare all'eliminazione di un blocco nel gruppo che viene poi nuovamente richiesto, causando un miss di questo tipo.

Le strategie che si possono adottare per diminuire il numero di miss sono:

 aumentare la quantità di cache, così da aumentare la quantità di dati immagazzinabili in essa==> Miss per capacità

 aumentare l'associatività, cioè il numero di linee per gruppo in cache. Tale tecnica però causa un incremento del tempo richiestoper localizzare il corretto gruppo in cui è presente l'istruzione o il dato cercato, e vi è inoltre la regola del 2:1, cioè una cache con N blocchi con associazione diretta ha quasi la stessa probabilità di miss di una cache con dimensione N/2 con associazione a 2 vie. ==> Miss per conflitto

 aumentare la dimensione di blocco, così da utilizzare la località spaziale

 separare la cache in cache istruzioni e cache dati, così da rendere indipendente la scrittura/lettura dei dati dalla singola lettura delle istruzioni

 utilizzare cache multilivello, cioè utilizzare una gerarchia di cache.Vi sarà un cache piccola, molto veloce e sullo stesso cip della CPU, detta cache on-chip, che permetterà ala CPU di accedere ai dati a "tempo 0"(cache L1). Vi è poi un'altracache(L2), più grande e leggermente più lenta, associare alla cache L1.Ve ne può anche essere una terza(L3),ancor

più grande e più lenta, associata alla L2. Le cache sono connesse tra di loro in modo indipendente dal bus di sistema, così da non pesare su di esso.

Nonostante l'aumento dei dispositivi tra CPU e memoria centrale, vi è unaumento prestazionalerispetto alla connessione direttadella L1alle memoria centrale, in quanto, grazie alla localizzazione spaziale, vi sarà un alto numero di hit dentro i dati nella L3, la quale, in caso di miss potrà inviare i dati sino alla L1 in un tempo estremamente ridotto rispetto a quello che farebbe la memoriacentrale.

 ottimizzare gli accessi alla cache mediante un compilatore che sia in grado di farlo. Tale compilatore deve essere in grado di: utilizzare in modo ottimale l'architettura della cache, ad esempio se multilivello; posizionamento accurato delle procedure ripetitive; incrementare la località spaziale mediante la fusione di vettori in strutture ottimali; incrementare la località spaziale mediante l'ottimizzazione di iterazioni annidate.

**Nel contesto di una pipeline, descrivere nel dettaglio la tecnica di predizione del salto utilizzando 2 bit di predizione.**

Con questa tecnica si cerca di prevedere durante il tempo di esecuzione se il salto sarà intrapreso oppure no.

Gli approcci statici di predizione sono saltare sempre, non saltare mai o stabilirlo in base al codice operativo.

Gli approcci dinamici sono bit taken/not taken o usando la tabella della storia dei salti.

Il bit taken/not taken indica se all’ultima esecuzione di quell’istruzione il salto è avvenuto oppure no (ricorda la volta precedente).

Nella predizione con due bit, se viene presa una decisione giusta si ripropone anche successivamente, se si rivela erronea si ripropone anche la volta successiva e se risulta ancora errata si cambia.

Es: previsto preso se è preso-> previsto ancora preso

Se è non preso->previsto ancora preso

se è preso->previsto preso

se è non preso->cambia previsione=non preso

**Spiegare cos’è il salto ritardato**

Il salto ritardato è 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.

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

La dipendenza dei dati è un problema che si verifica quando un’operazione dipende da qualche risultato della precedente.

Abbiamo 3 tipi di dipendenze:

-READ AFTER WRITE : i+1 legge prima che i abbia scritto

-WRITE AFTER WRITE: i+1 scrive prima che i abbia scritto

-WRITE AFTER READ: i+1 scrive prima che i abbia letto

Le possibili soluzioni sono:

--Introduzione di fasi non operative o stalli

--Risoluzione a livello di compilatore, che distanzia tra loro le operazioni che hanno dipendenze, cambiandone l’ordine, evitando il più possibile gli stalli. Esso sa riconoscere quando è possibile, e lo fa durante la compilazione subito dopo la traduzione

--Risoluzione a livello hardware, che riconosce le dipendenze e riordina le istruzioni in modo da ridurre i tempi di stallo, mantenendo la correttezza del programma.

--Data forwarding: individuata la dipendenza è possibile prelevare il dato direttamente dall’uscita della ALU e nel caso di un’architettura MIPS, sono previsti dei circuiti di bypass EX🡪EX o MEM🡪EX. Questa soluzione riduce notevolmente gli stalli di un’istruzione e quindi anche il numero di cic.0

li di clock di un’esecuzione completa.

**Si descrivano i formati delle istruzioni MIPS visti a lezione specificando per ogni formato la sua composizione tipica, i pregi e i difetti.**

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

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

Lo stadio ID è in grado di rilevare la dipendenza dei dati grazie ad un apposito circuito di identificazione delle dipendenze e gestione del dataforwarding. 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 ed i dati che devono essere mandati (inoltre manda segnali di controllo che regolano l'esecuzione e l'utilizzo dell'hardware per la memorizzazione ed il write back)

La Forwarding Unit è in grado di rilevare le dipendenze confrontando, mediante comparatori, i registri associati ai campi di input dell'istruzione che si trovano nello stadio ID con i registri target delle istruzioni precedente non ancora terminate. I comparatori possono leggere i registri associati ai campi di input e di output facendo riferimento al campo IR dei banchi di registri. L'istruzione che è nello stadio ID avrà le informazioni sui/sul registri/registro di input sempre nel banco IF/ID, mentre il registro dell'istruzione con cui fare il confronto sarà indicato nel banco ID/EX o EX/MEM o MEM/WB.

Se i due registri sono uguali e non vi è possibilità di dataforwarding viene generato uno stallo, altrimenti procede.

In particolare se:

 istruzione di input ha formato R i campi di input sono rs (IF/ID.IR[rs]) ed rt (IF/ID.IR[rt]) e verranno confrontati con:

o rd se l'istruzione precedente ha formato R

o rt se l'istruzione precedente ha formato I

 istruzione di input ha formato I il campo di input è rs (IF/ID.IR[rs]) e verrà confrontato con:

o rd se l'istruzione precedente ha formato R

o rt se l'istruzione precedente ha formato I

**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.**

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). Inefficacie 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, poichè sarebbe maggiore la probabilità di trovarle nel buffer.

**Spiegare le motivazioni di base dell’architettura CISC**

CISC sta per complex instruction set computer ed è un’architettura nata con lo scopo di migliorare l’efficienza dell’esecuzione dei programmi, facilitare il lavoro del compilatore e supportare linguaggi ad alto livello più complessi.

Interviene sulla questione dell’efficienza implementando le operazioni complesse non più con una sequenza di istruzioni ma mettendo a disposizione istruzioni singole in grado di svolgere le stesse attività.

Utilizza un set di istruzioni molto ampio, prevalgono le implementazioni in hardware poiché meno costoso del software e dispone di svariati modi di indirizzamento.

L'architettura CISC nacque per diversi motivi, di cui i più importanti sono:

 facilitare la scrittura del compilatore

 supportare i linguaggi ad alto livello sempre più complessi

 migliorare l'efficienza dell'esecuzione, in quanto essendo state implementate operazioni più complesse tramite microcodice, si dovrebbe aver avuto un incremento prestazionale non dovendo fare una successione di istruzioni primitive per eseguire l'istruzione complessa.

 si pensava di poter ottenere programmi più piccoli, che occupassero meno memoria e fossero eseguiti più velocemente

 facilitare il lavoro del programmatore a discapito dell'aumento del costo dell'hardware. Il costo del software è molto maggiore del costo dell'hardware, in quanto l'hardware può essere ammortizzato nel tempo, mentre vengono spese migliaia di ore uomo su uno stesso computer.

 permettere grande flessibilità mettendo a disposizione un ampio set di istruzioni e svariati metodi di indirizzamento.

**Si metta a confronto criticamente il modo in cui un’architettura RISC utilizza l’ampio banco di registri a sua disposizione rispetto alla gestione di una cache**

L’architettura RISC utilizza l’ampio banco di registri per conservare le variabili (prevalentemente scalari locali) che hanno un’alta probabilità di essere utilizzate con maggior frequenza. Sotto questo punto di vista il banco dei registri assomiglia molto alla memoria cache, sbbene sia molto più veloce. Ci sono però alcune sostanziali differenze:

1. Il BR contiene tutti gli scalari locali delle ultime N-1 procedure attivate e la cache contiene gli scalari locali usati di recente.
2. Le variabili globali sono assegnate dal compilatore per quanto riguarda il BR, mentre la cache contiene solo quelle usate di recente.
3. Mentre il BR contiene solo le variabili in uso, la cache importa un blocco di dati che potrebbe non essere utilizzato.
4. Il trasferimento dati tra registri e memoria è determinato in base alla profondità di annidamento delle procedure, mentre per la cache il salvataggio e il rimpiazzo è determinato in base all’algoritmo di sostituzione adottato.
5. Per accedere ad uno scalare locale in un BR viene utilizzato l’indirizzamento a registro, che è molto semplice e veloce.

Per quanto riguarda la cache invece, l’indirizzamento è molto più lento. La maggior parte delle cache infatti sono set-associative, il che comporta delle operazioni di confronto con il set e il tag per determinare la presenza o no di un dato in cache.

Anche nel caso della cache a indirizzamento diretto o fully-associative, questi modi sono comunque indirizzamenti a memoria che sono sempre più lenti di un indirizzamento a registro.

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

L’obiettivo del compilatore è mantenere gli operandi necessari nei registri per la maggior parte del tempo e minimizzare il numero di accessi alla memoria.

Il suo compito è eseguire un’approfondita analisi del programma sottopostogli e mappare ogni registro simbolico (o virtuale) a cui è stata assegnata una variabile, in un registro fisico del processore.

Più registri simbolici possono esser mappati su uno stesso registro reale se il loro uso non si sovrappone temporalmente.

Se i registri reali non sono sufficienti per contenere tutte le variabili riferite in un certo intervallo di tempo, allora queste ultime vengono assegnate a locazioni di memoria.

L’essenza dell’ottimizzazione è quella di decidere quali variabili debbano essere assegnate ai registri in un certo punto del programma. Il mapping delle variabili è equivalente alla risoluzione del problema di colorazione di un grafo. In questo problema dato un grafo costituito da nodi connessi ad archi, si vuole assegnare un colore per ogni nodo in modo tale che:

* Nodi adiacenti connessi da archi abbiamo colori diversi
* Usare il minor numero possibile di colori

I nodi rappresentano i registri virtuali e i colori i registri reali.

Essendovi a disposizione n registri reali, il grafo dovrà essere colorato con al massimo n colori. Se vi sono nodi che non potranno essere colorati, allora verranno copiati in memoria.

**Spiegare in dettaglio come un’architettura RISC possa trattare efficientemente la chiamata annidata di procedure**

Dall'osservazione che le 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 che memorizzano il contenuto delle variabili locali della procedura e registri temporanei che gestiscono il ritorno della procedura.

I registri temporanei di un gruppo si sovrappongono perfettamente con quelli che contengono i 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.

Nel buffer circolare, quando avviene una chiamata il puntatore alla finestra corrente (CWP) viene aggiornato per farlo puntare alla finestra attiva. Se si esaurisce la capacità del buffer, cioè tutte le 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 quindi 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.

**Si spieghi in dettaglio lo schema per realizzare la moltiplicazione tra numeri reali dello standard IEEE 754**

1. Controllo dello zero, se uno dei numeri è zero il risultato sarà zero, quindi non ha senso procedere. Se sono entrambi diversi da zero si continua.

1. Somma degli esponenti.
2. Essendo entrambi gli esponenti polarizzati, la loro somma ha raddoppiato a polarizzazione in quanto è stata considerata due volte quindi va sottratta.
3. Moltiplicazione degli operandi
4. Normalizzazione del risultato
5. Se il prodotto supera la lunghezza del formato massimo consentito dalla virgola mobile, si arrotonda tramite l’utilizzo dei formati estesi per i risultati intermedi. Tale arrotondamento può avvenire per: arrotondamento al più vicino, per eccesso, per troncamento.

**Si spieghi in dettaglio l’hardware adottato per realizzare la somma e sottrazione di numeri interi rappresentati in complemento a due.**

L’elemento centrale dell’hardware è il sommatore binario (adder) al quale vengono forniti gli addendi o sottraendo e minuendo.

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.

**Spiegare in dettaglio la divisione fra numeri reali secondo lo standard IEEE 754**

1. Controllo dello zero. Se il divisore è zero, viene inviata una segnalazione d’errore. Se il dividendo è zero si ottiene un risultato nullo.
2. L’esponente del divisore viene sottratto dall’esponente del dividendo.
3. L’ultimo passaggio ha rimosso la polarizzazione e deve essere risommata.
4. Vengono eseguiti dei controlli sull’underflow o sull’overflow dell’esponente.
5. Divisione dei significandi
6. Normalizzazione
7. Arrotondamento

Nell'effettuare la divisine tra due numeri floating point secondo standard IEEE 754 bisogna seguire i seguenti passaggi:

 controllo se uno o entrambi gli operandi sono zero.

Se il divisore è zero verrà dato errore, quindi Non A Number, se il dividendo è zero ed il divisore è diverso da zero, verrà dato output zero. In entrambi questi casi il processo di divisione termina.

 l'esponente del divisore viene sottratto all'esponente del dividendo

 nell'esponente risultate dal passo precedente è stata sottratta la polarizzazione, in quanto era stata applicata ad entrambi gli esponenti ed avendone fatto la differenza la polarizzazione è stata persa. La polarizzazione va quindi nuovamente ripristinata. (In pratica all'esponente risultate dal passo precedente viene sottratto il valore 2k-1-1)

Se non ci sono underflow od overflow di esponente la procedura continua.

 avviene la divisione degli operandi

 il quoziente della divisione viene normalizzato, cioè l'esponente è aggiustato in modo che il bit più significativo della mantissa sia 1.

 se il quoziente risulta essere in un registro più lungo del formato massimo consentito della virgola mobile, grazie all'utilizzo dei formati estesi per i risultati intermedi, è necessario arrotondarlo. Tale arrotondamento può avvenire per: arrotondamento al più vicino, per eccesso, per troncamento.

**Si spieghi in dettaglio la codifica in complemento a due dei numeri interi. Si discutano poi i problemi legati alla realizzazione della moltiplicazione di due interi rappresentati in complemento a 2, esemplificando tali problemi su un caso concreto di moltiplicazione.**

Attraverso la 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

Il segno del numero viene indicato mediante il bit più a sinistra, il qauel rappresenta – se è a 1 e + se è a 0.

La rappresentazione per i numeri positivi è identica a quella in modulo e segno, entrambe con lo zero a sinistra.

I numeri positivi rappresentabili vanno da 0 a 2^(n-1)-1, i negativi da -1 a -2^(n-1).

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 complementati.

Nella moltiplicazione di due interi rappresentati in complemento a due nascono problemi legati al bit più significativo, cioè al bit di segno. 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.

**Si spieghi in dettaglio la rappresentazione dei numeri reali secondo lo standard IEEE 754.**

Lo standard IEEE 754 è lo standard per i numeri in virgola mobile.

Abbiamo due formati: formato singolo a 32 bit e doppio a 64 bit.

Il formato singolo a 32 bit è composto da 1 bit di segno, 8 bit per l’esponente e 23 per la mantissa.

Il formato doppio a 64 bit è composto da 1 bit di segno, 11 bit per l’esponente e 52 per la mantissa.

Inoltre abbiamo altri due formati estesi (singolo e doppio) che includono bit aggiuntivi per l’esponente (intervallo esteso) e per la mantissa (precisione estesa) per risultati intermedi.

Nella IEEE 754 la rappresentazione dell’esponente è polarizzata, e per ottenere suo il vero valore bisogna sottrarre al campo un valore detto polarizzazione che è pari a 2^(k-1)-1

Per esempio nel caso del formato singolo, con 8 bit per l’esponente, è 127.

Per semplificare le operazioni sui numeri in virgola mobile viene effettuata la normalizzazione.

In un numero normalizzato la cifra più significativa (più a sinistra) del significando è uguale a 1.

Come convenzione deve esserci un solo bit a sinistra della virgola.

E poiché il bit più significativo è sempre 1 non è necessario memorizzarlo.

**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 calore da sommare ad A per ottenere l’indirizzo.

**Nel contesto di una pipeline, descrivere nel dettaglio la tecnica del data-forwarding: a cosa serve? Come funziona? Di che supporto hardware ha bisogno?**

Il data forwarding è una delle soluzioni utilizzate per superare le dipendenze dai dati dai dati che si possono verificare tra le istruzioni in una pipeline.

Se viene individuato una dipendenza dai dati, il data forwarding, se il tipo di dipendenza lo permette, 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 potenzialmente anche da altre unità (dipende dall'architettura), 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 dataforward può essere implementato in diversi modi che dipendono dall'architettura su cui lo si implementa.

Un esempio pratico di data forward lo troviamo nell'architettura MIPS in cui vi è un apposito circuito di identificazione delle dipendenze e gestione del dataforwarding. 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 ed i dati che devono essere mandati (inoltre manda segnali di controllo che regolano l'esecuzione e l'utilizzo dell'hardware per la memorizzazione ed il write back)

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 EX a MEM.

**Formato delle istruzioni**

Il formato delle istruzioni descrive i campi dell'istruzione, la sua lunghezza ed numero di indirizzi. In qualsiasi formato è incluso il codice operativo, che discrimina quale operazione fare, ed zero, uno o più operandi in modo implicito o esplicito.

--- introduzione generale ---

Il formato delle 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 una lunghezza che dipende dal numero di operandi e nel campo con l'opcode devo anche specificare il numero di operandi. Permettono una grande flessibilità, ma incrementano notevolmente la complessità.

 ibrida, ho diversi formati con lunghezza fissa ma diversa

La lunghezza è data da un compromesso tra repertorio di 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 il numero di accessi), complessità della CPU (CPU con più istruzioni necessita di più bit per l'opcode) e la velocità richiesta della CPU(se la CPU utilizza la pipeline).

L'allocazione dei bit nei campi d'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 (solo alcune architetture li posseggono), intervallo di indirizzi da rendere disponibile, granularità d'indirizzamento (byte o parola, a volte può essere utile il byte anche se più costoso nel numero di accessi).

---

Prima di inoltrasi nella descrizione dei formati è utile precisare degli svantaggi e vantaggi generali:

 Una grande varietà di istruzioni e di lunghezze differenti permettono l'utilizzo di un 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(rendendola anche più costosa), rende difficile il fetch e globalmente diminuisce la reattività sulle operazione più frequentemente utilizzate.

Tale caratteristiche si avvicinano alla filosofia CISC.

 Un numero fisso di formati ha come problemi la mancanza di ortogonalità(cioè operandi indipendenti dal codice operativo) tra opcode e metodo d'indirizzamento ed un numero limitato di operandi utilizzabili per operazione. In generale vi è una mancanza di flessibilità. La CPU è però più semplice da realizzare, generalmente più, permette il caricamento dell'istruzione in modo uniforme veloce ed una dimensione fissa delle istruzioni favorisce l'uso della pipeline. Tale caratteristiche si avvicinano alla filosofia RISC.

I possibili formati di codifica di una istruzione sono:

 PDP-8: architettura molto vecchia in cui era disponibile un solo registro, l'accumulatore. Nonostante le forti limitazioni vi è un indirizzamento abbastanza flessibile. Le 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: innovativo rispetto al PDP-8: 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. (scelta progettuale)

 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 varabile e formati diversi, in quanto l'opcode può stare su un byte o su due. 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.

**Si discuta nel dettaglio in cosa consista il formato variabile per le istruzioni. Se possibile, dare esempi di formati variabili.**

Un formato variabile per le istruzioni è un formato che permette una grande varietà di istruzioni ed una grande flessibilità.

Tali istruzioni hanno lunghezze variabili, cioè ogni istruzione ha una lunghezza che dipende dal numero di operandi ed ha un campo opcode generalmente espanso, così da permettere la codifica di un vasto numero di operazioni e metodi di indirizzamento ed inoltre il campo nel campo opcode viene anche specificare il numero di operandi presenti nell'istruzione. Una istruzione con formato variabile può inoltre avere un numero variabile di campi di tipo diverso, come ad esempio nel PENTIUM.

L'utilizzo di questo tipo di istruzioni complica notevolmente la CPU, la rende più costosa, rende difficile il fetch e globalmente diminuisce la reattività sulle operazione più frequentemente utilizzate.

Tale caratteristiche si avvicinano alla filosofia CISC.

Due esempi di formati variabili delle istruzioni li troviamo storicamente nel:

 VAX: un sistema estremamente variabile con lunghezza varabile e formati diversi, in quanto l'opcode può stare su un byte o su due. 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.

**Si descriva sinteticamente l'implementazioni delle istruzioni attraverso la tecnica della microprogrammazione. Si dica se questa tecnica viene utilizzata per i processori CISC o RISC, motivare la risposta.**

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.

**Descrivere i possibili approcci per trattare l'indirizzo di ritorno da una chiamata di procedura.**

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 ritono e la seconda la vera istruzione da eseguire. Volendo schematizzare:

 X = PC + Δ

 PC = X + 1

Questi 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.

**Si descriva sinteticamente l'implementazione delle istruzioni attraverso la tecnica della microprogrammazione. Si dica se questa tecnica viene utilizzata per i processori CISC o RISC, motivare la risposta.**

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 microoperazioni 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 microoperazione 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.

**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, dovuta all'aumento delle complessità, ed all'aumentare della frequenza di clock 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 adarchitetture 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).

**Si descrivano le possibili alternative di organizzazione per un processore multicore**.

L'organizzazione di un processore multicore dipende da:

 il numero di core per cip

 la tipologia di core, cioè sei i singoli core sono superscalari o sono multhithreading simultaneo (SMT). I 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, la cui caratteristica divide i multicore in 4 macrogruppi:

o Cache L1 dedicata: ogni core ha la propria cache L1 dedicata, la quale è suddivisa tra cache dati e cache istruzioni.

o Cache L2 dedicata: ogni core ha la propria cache L1 ed L2 dedicata.

o Cache L2 condivisa: ogni core ha la propria cache L1 dedicata, ma vi è una cache L2 condivisa tra tutti i core.

o Cache L3 condivisa: ogni core ha la propria cache L1 ed L2 dedicata, ma vi è una cache L3 condivisa tra tutti i core.

---- Aggiunta ----

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 tra dentro al processore sono più facili da realizzare

5. Il problema della coerenza dei dati viene confinata nella cache L1.

ed i seguenti svantaggi:

1. l'accesso a cache L2 dedicata è più rapido, utile per programmi con alta località spaziale

L'utilizzo una cache L3 condivisa può migliorare ulteriormente le prestazioni.