PARTE 2

9 - ARITMETICA DEL CALCOLATORE

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

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 lo zero ha 2 rappresentazioni e per eseguire correttamente somme e sottrazioni occorre considerare i segni dei due numeri e i loro moduli separatamente.
Attraverso rappresentazione in complemento a due, con n bit a disposizione possiamo rappresentare tutti i numeri interi da 2^(n1) a +2^(n1)1. Il segno viene indicato mediante il b il più a sinistra, il quale rappresenta – se è a 1 e + se è a 0. La codifica in complemento a 2 associa il seguente valore a stringhe di n bit: $-2^{n-1}a_{n-1} + \sum_{i=0}^{n-2} 2^i\, a_i$.
Quindi la rappresentazione per i numeri positivi è identica a quella in modulo e segno mentre per calcolare un numero negativo vengono utilizzati due metodi: si calcola la sua versione positiva in binario e si complementa a 2 (complemento 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.

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^(n1) a +2^(n1)1. La codifica in complemento a 2 associa il seguente valore a stringhe di n bit: $-2^{n-1}a_{n-1} + \sum_{i=0}^{n-2} 2^i\, a_i$. Il segno viene indicato mediante il bit più a sinistra, il quale rappresenta – se è a 1 e + se è a 0. La rappresentazione per i numeri positivi è identica a quella in modulo e segno mentre per calcolare un numero negativo vengono utilizzati due metodi: si calcola la sua versione positiva in binario e si complementa a 2 (complemento 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. 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. Un esempio è -5 * -3 = -113. La soluzione adottata è l'algoritmo di Booth.

Hardware adottato per 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 i n 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 rappresentazione dei numeri reali secondo lo standard IEEE 754.

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).

Spiegare nel dettaglio lo schema per realizzare la moltiplicazione tra numeri reali dello standard IEEE 754

Se uno dei due operandi è 0 il risultato è 0. Il passo successivo è sommare gli esponenti ma, visto che essi sono memorizzati in forma polarizzata, la loro somma raddoppierebbe la polarizzazione quindi viene sottratto il valore della. 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 e seguita come per gli interi. Concluso il calcolo del prodotto, il risultato viene normalizzato e arrotondato (La normalizzazione potrebbe comportare un overflow dell'esponente).

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

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

1. 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.
2. l'esponente del divisore viene sottratto all'esponente del dividendo e viene risommata la polarizzazione. Se non ci sono underflow od overflow di esponente la procedura continua.
3. avviene la divisione degli operandi
4. il quoziente della divisione viene normalizzato, cioè l'esponente è aggiustato in modo che il bit più significativo della mantissa sia 1.
5. 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.

10 - LINGUAGGI MACCHINA

Possibili approcci per trattare l'indirizzo di ritorno di 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 e carica l'indirizzo contenuto nel program counter.
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. Questo metodo consente procedure annidate, ma non rientranti, cioè non si possono avere più chiamate aperte contemporaneamente per una stessa procedura
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 rientranti.

11 - MODI DI INDIRIZZAMENTO E FORMATI

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.

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 2^k con k la lunghezza del campo indirizzo.
Lo svantaggio di tale metodo è che esso richiede due accessi in memoria per ottenere l'operando: uno per estrarne l'indirizzo e uno per ottenerne il valore.
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.
Tale tecnica richiede che l'istruzione abbia due campi indirizzo: il campo A, che contiene il valore di un campo indirizzo e viene usato direttamente, e il campo R, che può essere implicito in base al codice operativo, che punta a un registro che contiene l'indirizzo di un valore da sommare ad A per ottenere l'indirizzo.
I 3 più comuni utilizzi dell'indirizzamento con spiazzamento sono:

- Indirizzamento relativo: il registro implicitamente referenziato è il Program Counter
- Indirizzamento registro-base: il registro referenziato contiene un indirizzo di memoria e il campo indirizzo contiene uno spiazzamento
- Indicizzazione: il campo indirizzo rappresenta un indirizzo in memoria centrale e il registro rappresentato contiene uno spiazzamento positivo da tale indirizzo. Fornisce un meccanismo efficiente per eseguire operazioni iterative.

La pila è una sequenza lineare di locazioni di memoria riservate. Associato alla pila troviamo un puntatore (contenuto nel registro SP) il cui valore è l'indirizzo della cima della pila. Quindi l'indirizzamento a pila è una forma di indirizzamento a registro indiretto implicito.

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

Il formato variabile per le istruzioni permette di definire per uno stesso calcolatore istruzioni con formati diversi, cioè on diverse lunghezze e diposizioni di bit per definire l'opcode e i relativi operandi e metodi di indirizzamento. Questo permette una grande varietà di istruzioni, una grande flessibilità di indirizzamento e libertà nell'uso degli operandi. Inoltre tali istruzioni possono avere lunghezze variabili in modo da permettere ancor più operazioni, operandi e i metodi di indirizzamento usati. 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 operazioni più frequentemente utilizzate. Tale caratteristiche si avvicinano alla filosofia CISC.
Due esempi di formati variabili delle istruzioni li troviamo storicamente nel:
• VAX: un sistema con lunghezza varabile e formati diversi, in cui sono necessari uno o due byte solo per l'opcode perché esistono più di 300 operazioni che hanno fino a 6 operandi. È un sistema molto flessibile e potente che facilita il lavoro di programmatori e compilatori, ma al costo di un sistema molto complesso.
• PDP11: Le istruzioni sono lunghe 16, 32 o 48 bit e la lunghezza dell'opcode varia da 4 a 16 bit.

Si descrivano i possibili formati di codifica di un'istruzione, specificando per ogni formato la sua composizione tipica, i pregi e i 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 sui 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.

12 - STRUTTURA E FUNZIONE DEL PROCESSORE

Nel contesto di una pipeline, descrivere in dettaglio la tecnica del buffer circolare, spiegando quale problema risolve.

Un buffer circolare è una memoria piccola e molto veloce, gestita nella fase di fetch delle istruzioni, che contiene le ultime n istruzioni prelevate. Se occorre effettuare un salto, l'hardware prima controlla se la destinazione si trova nel buffer. In caso affermativo, la successiva istruzione viene prelevata dal buffer. Questa tecnica presenta tre vantaggi:

1. Anticipando il fetch, il buffer circolare conterrà alcune istruzioni successive all'indirizzo di prelievo dell'istruzione corrente quindi le prossime istruzioni saranno disponibile senza dover attendere il tempo di accesso alla memoria
2. Se si verifica un salto a una destinazione che si trova poche celle più avanti la destinazione si trova già nel buffer
3. Questa strategia è particolarmente adatta per il trattamento di cicli perché se il buffer può contenere tutte le istruzioni allora esse vengono prelevate dalla memoria solo la prima volta

Questo sistema viene usato per risolvere le dipendenze da controllo che si verificano quando la pipeline intraprende decisione errata su una predizione di salto e di conseguenza porta nella pipeline istruzioni che devono essere successivamente eliminate.

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.

La dipendenza dei dati è un problema che 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.
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 cicli di clock di un'esecuzione completa.

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.

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

Per prevedere se un salto verrà intrapreso possono essere utilizzate varie tecniche. È 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.

13 - PROCESSORI RISC

Spiegare nel dettaglio come una 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.

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.

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, sebbene 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. 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.
3. Il banco dei registri fa un uso inefficiente dello spazio quando non tutte le procedure richiedono l'intera dimensione della finestra ad esse allocata, mentre la cache è inefficiente perché importa i dati in blocchi e non tutto il blocco potrebbe essere usato.
4. La cache è in grado di trattare variabili locali e globali e scoprire dinamicamente quali variabili globali sono maggiormente utilizzate e contenerle in memoria, invece per il banco dei registri questo compito è demandato al compilatore, tuttavia risulta molto difficile.
5. 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.

6. 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.

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. L'essenza dell'ottimizzazione è quella di decidere quali variabili debbano essere assegnate ai registri in un certo punto del programma e si ottiene risolvendo il problema di colorazione di un grafo i cui nodi, che rappresentano i registri virtuali, sono connessi se i corrispondenti registri si sovrappongono temporalmente e 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

Essendovi a disposizione n registri reali, il grafo dovrà essere colorato con al massimo n colori e i rimanenti allora verranno copiati in memoria.

Motivazioni di base dell'architettura CISC.

L'architettura CISC nacque per diversi motivi, di cui i più importanti sono:
• semplificare la scrittura del compilatore mettendo a disposizione istruzioni macchina simili ai comandi di alto livello
• 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, cioè che occupassero meno memoria, e fossero eseguiti più velocemente
• facilitare il lavoro del programmatore a discapito dell'aumento del costo dell'hardware.
• permettere grande flessibilità mettendo a disposizione un ampio set di istruzioni e svariati metodi di indirizzamento.

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.

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

La MIPS prevede 3 formati di istruzione:

- registro (R), utilizzato per le operazioni logico-aritmetiche
  Il formato R ha i campi: op rs rt rd shamt funct
  op rappresenta il codice operativo
  rs e rt contengono gli operandi
  rd contiene il registro di destinazione
  shamt la quantità di shift e funct il tipo di operazione.
- 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.
  Il formato I ha i campi: op rs rt address
  rs contiene il contenuto del registro base
  rt contiene il contenuto del registro che deve essere caricato dalla memoria o in memoria
  address il dato immediato.
- jump (J), per le istruzioni di salto.
  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:
  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:
  rd se l'istruzione precedente ha formato R o rt se l'istruzione precedente ha formato I

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.

15 - MICROPROGRAMMAZIONE

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.

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.

18 - CALCOLATORI MULTICORE

Discutere le motivazioni alla base dei processori multicore

I microprocessori hanno visto una crescita esponenziale delle prestazioni grazie al miglioramento dell'organizzazione e dall'incremento della 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 s i 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 (se 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.