**Parte I - Capitolo 4 - Il Processore**

Come si è visto in precedenza le prestazioni di un calcolatore dipendono dal numero di istruzioni, dalla durata del ciclo di clock e dal numero di cicli di clock per istruzione (CPI); tutti questi fattori dipendono a loro volta dalla particolare implementazione del processore.

Si esamina ora un'implementazione che comprende le istruzioni di base del MIPS quali le istruzioni di riferimento alla memoria (load e store), le istruzioni logico-aritmetiche e le istruzioni di salto condizionato e incondizionato. Tutte queste istruzioni possono essere riassunte attraverso due passi: per prima cosa si invia il contenuto del program counter alla memoria che contiene il programma e si preleva l'istruzione dalla memoria (***fetch***); il passo successivo consiste nel leggere il contenuto dei registri. Dopo aver effettuato questi due passi le azioni richieste per completare l'esecuzione dipendono dalla tipologia delle istruzioni. Eccetto i salti incondizionati, tutti i tipi di istruzioni utilizzano la ***ALU*** (***Arithmetic-Logic Unit***). Oltretutto bisogna introdurre un circuito logico che funga da selettore di dati, in quanto nella ALU sono connesse differenti sorgenti; questo compito viene svolto da un dispositivo detto ***multiplexer***. Nella progettazione di un processore possiamo distinguere tra due tipi di elementi: gli ***elementi combinatori*** sono quegli elementi che operano sui dati e che sono in grado di eseguire operazioni su di essi; gli ***elementi di stato*** sono elementi che hanno la possibilità di registrare dati in memoria. Un elemento di stato (es. flip flop di tipo D) possiede almeno due ingressi (un valore e il clock) e un'uscita e sono anche detti *sequenziali* perchè l'uscita dipende sia dagli ingressi che dal valore al suo interno. Altro punto importante è la metodologia di temporizzazione che definisce quando possono essere scritti o letti i segnali. Per semplicità useremo il tipo di temporizzazione sensibile ai fronti detta ***edge triggered*** che garantisce che il valore all'interno di un elemento logico venga aggiornato solo in corrispondenza di un segnale di clock.

Analizziamo gli elementi dell'unità di elaborazione (datapath elements). Prima di tutto abbiamo bisogno di un'unità di memoria in cui salvare le istruzioni del programma che sia in grado di fornire in uscita l'istruzione associata all'indirizzo dato in ingresso. L'altro elemento è il ***program counter*** che è un registro utilizzato per la memorizzazione dell'indirizzo dell'istruzione corrente. C'è inoltre bisogno di un sommatore per incrementare il program counter e ottenere l'indirizzo dell'istruzione successiva. Consideriamo ora le istruzioni di tipo R alla base del MIPS. I registri universali di 32bit sono raccolti in una struttura detta ***register file***, un insieme di registri in cui ciascun registro può essere letto o scritto specificando il numero ad esso associato. Avremo inoltre bisogno di una ALU per svolgere le operazioni sui registri.

- Vedere sulle slide come si implementano le varie funzioni in MIPS

Il ***pipelining*** è una tecnica di implementazione hardware di un insieme di istruzioni che prevede la sovrapposizione temporale dell'esecuzione di diverse istruzioni; il pipelining è utilizzato oggi in tutti i calcolatori. In questa tecnica non viene ridotto il tempo di esecuzione per portare a termine ogni stato, ma viene ridotto il throughput in modo da ridurre il tempo complessivo del lavoro. La pipelininig applicata ai processori prevede cinque diversi stati per l'esecuzione di diverse istruzioni: *fetch dell'istruzione* (IF, Instruction Fetch), *lettura dei registri* (ID, Instruction Decode), *esecuzione di un'operazione* (EX, Execution Operation), *accesso a un operando della memoria dati* (MEM, Access memory operand), *scrittura del risultato in un registro* (WB, Write result back to register). In condizioni ideali con un numero molto grande di istruzioni, l'incremento della velocità dovuto alla pipeline è pari al numero dei suoi stadi. Un problema dovuto alla tecnica del pipeline è detta ***hazard*** (o ***criticità***), cioè situazioni in cui l'istruzione successiva non può essere eseguita nel ciclo di clock immediatamente successivo e possono essere di diversi tipi.

Gli ***hazard strutturali*** si verificano quando le risorse hardware presenti, non sono in grado di supportare la combinazione di istruzioni che vorrebbe eseguire nello stesso ciclo di clock (es. una delle risorse che serve è occupata). Gli ***hazard sui dati*** si verificano quando la pipeline deve essere messa in stallo perchè uno stadio deve attendere che termini l'elaborazione in un altro stadio della pipeline (es. mi servono i risultati dell'istruzione precedente, ma questi non sono ancora pronti). Gli ***hazard sul controllo*** o ***hazard sui salti condizionati*** che avvengono quando bisogna prendere una decisione in funzione del risultato dell'esecuzione di un'istruzione e altre istruzioni sono già state avviate all'esecuzione.

Quando si suppone che un salto condizionato non debba essere eseguito si realizza una forma rozza di ***predizione del salto***. Nel caso in cui la predizione si riveli errata, bisogna eliminare le istruzioni contenute nella pipeline. Con pipeline più profonde il costo dei salti condizionati (misurati in numero di cicli di clock) aumenta. Un possibile approccio per la predizione dei salti condizionati consiste nel verificare se l'ultima volta che un'istruzione di salto è stata eseguita il salto sia effettivamente eseguito: in caso positivo vengono caricate le istruzioni a partire da quella presente nell'indirizzo di destinazione del salto. Questa tecnica viene chiamata ***predizione dinamica dei salti***. Un modo per implementare questa strategia consiste nell'utilizzare un buffer di predizione dei salti (***brench prediction buffer***), detto anche tabella della storia dei salti che è una piccola memoria indicizzata che contiene un bit che indica se il salto era stato eseguito o meno nell'ultima esecuzione. La tecnica della predizione è un suggerimento che si spera risulti corretto. Se questo si risulta sbagliato, le istruzioni caricate erroneamente saranno eliminate, verrà invertito il bit di predizione e l'esecuzione ripartirà dal prelevamento dell'istruzione corretta.

L'unità di controllo di un processore è la parte più difficile da far funzionare correttamente ed è anche la più difficile da rendere veloce. Una delle sfide maggiori è rappresentata dalla gestione delle ***eccezioni*** e dagli ***interrupt***. Molte architetture e molti ricercatori non fanno distinzione tra interrupt ed eccezioni e utilizzano spesso il termine interrupt per indicare entrambi i tipi di eventi (es. Intel x86). Con riferimento alle convenzioni MIPS utilizzeremo il termine eccezione riferendoci a un qualsiasi cambiamento non previsto del flusso di controllo, senza distinguere se questo abbia cause interne o esterne al processore; utilizzeremo il termine interrupt per riferirci ai soli eventi che hanno cause esterne. Questi eventi possono essere generati dall'esecuzione di un'istruzione non valida o dall'overflow aritmetico. L'operazione principale che il processore deve compiere consiste nel salvare l'indirizzo dell'istruzione che ha generato l'eccezione nel ***program counter delle eccezioni*** (***EPC***, ***Exception Program Counter***) e trasferire il controllo a un indirizzo specifico del sistema operativo. Il sistema operativo, a questo punto, potrà intraprendere le azioni più opportune come servire un certo servizio al programma utente, eseguire in risposta dell'overflow un'azione predeterminata, terminare l'esecuzione del programma segnalando un errore. Dopodichè il sistema operativo può terminare il programma o riprenderne l'esecuzione utilizzando l'EPC per determinare il punto da cui ripartire. Per poter gestire al meglio un'eccezione bisogna conoscerne la causa, oltre a sapere quale istruzione l'ha generata. Ci sono due metodi per comunicare la causa: l'architettura MIPS utilizza un registro dedicato (***registro Causa***) contenente un campo che indica la causa dell'eccezione; un metodo alternativo consiste nell'adottare ***interrupt vettorizzati*** nei quali l'indirizzo a cui si deve trasferire il controllo viene determinato dalla causa dell'eccezione stessa.

La piepline sfrutta il parallelismo potenziale esistente tra le istruzioni. Questa forma di parallelismo è detta ***parallelismo a livello di istruzione*** (***ILP***, ***Instruction-Level Parallelism***), incrementabile attraverso due metodi. Il primo metodo consiste nell'aumentare la profondità della pipeline per sovrapporre più istruzioni. In questo modo la quantità di parallelismo è più alta, poichè più istruzioni vengono sovrapposte nel tempo e le prestazioni sono potenzialmente migliori dato che il ciclo di clock può essere ridotto. Un altro approccio consiste nel replicare i componenti interni del calcolatore in modo tale che sia possibile lanciare l'esecuzione di più istruzioni in ogni stadio della pipeline. Il nome generico di questa tecnica è ***esecuzione parallela*** (***multiple-issue***). Tuttavia, anche se la pipeline può essere resa più veloce sfruttando il parallelismo, ci sono molti vincoli sulle istruzioni che possono essere eseguite in parallelo e su cosa fare quando di verificano delle dipendenze. Esistono due modalità principali per realizzare processori a esecuzione parallela. La principale differenza consiste nel modo in cui viene suddiviso il lavoro tra compilatore e hardware: si parla di ***parallelizzazione statica dell'esecuzione*** (***static multiple issue***) se le decisioni vengono prese durante la compilazione, o di ***parallelizzazione dinamica dell'esecuzione*** (***dynamic multiple issue***) se le decisioni vengono prese durante l'esecuzione.

Uno dei metodi più importanti per scoprire e sfruttare al massimo il parallelismo è la ***speculazione*** che è un approccio che permette al compilatore o al processore di "indovinare" le caratteristiche di un'istruzione in modo da permettere l'inizio dell'esecuzione di altre istruzioni che dipendono da quell'istruzione. Il problema principale è che la speculazione può rivelarsi sbagliata, quindi il meccanismo deve prevedere anche la verifica della correttezza per tornare indietro o eliminare gli effetti delle istruzioni eseguite da quest'approccio. La speculazione può essere fatta dal compilatore (es. riordinare le istruzioni) o dal processore (es. stesse del compilatore, ma fatte durante l'esecuzione). Per quanto riguarda la speculazione software, il compilatore inserisce istruzioni aggiuntive che controllano l'accuratezza della speculazione e fornisce una procedura di riparazione da utilizzare quando si verifica un errore. Nel caso di speculazione hardware, il processore memorizza e mantiene in buffer dedicati i risultati dell'esecuzione finchè non capisce se la speculazione è corretta o meno. Se la speculazione è corretta l'esecuzione viene completata scrivendo i risultati nei registri o nella memoria; se la speculazione è errata, l'hardware elimina il contenuto di questi buffer e riprende l'esecuzione con la sequenza corretta di istruzioni.