**PROCESSORI**

**Che effetto ha l’istruzione più lunga sulla pipe?**

Determina la durata del ciclo di clock.

Tutti gli stadi della pipeline richiedono un ciclo di clock per cui il periodo di clock deve essere abbastanza lungo da contenere l’operazione più lenta.

**HAZARD**

**Cosa sono gli Hazard?**

Hazard sono situazioni in cui non posso prevedere il risultato dell’operazione e quindi e quindi sono costretto a bloccare la pipeline perché devo aspettare qualcosa che non è ancora pronto.

Un Hazard è una situazione che si verifica quando non posso eseguire l’istruzione successiva nel ciclo di clock immediatamente successivo

Gli Hazard posso essere di tre tipi Hazard strutturali, Hazard sui dati, Hazard sul controllo.

**Hazard strutturali** si verificano quando una risorsa di cui ho bisogno per eseguire uno stadio della pipe è occupato e quindi devo attendere. Nel caso in cui nel MIPS avessimo avuto solo una memoria per sia per le istruzioni che per i dati l’architettura con le pipeline presenterebbe un **Hazard strutturale**.

Se avessi avuto una sola memoria a cui accedere non potevo fare il fetch se sto accedendo alla memoria per fare una load o una store perché la memoria una cosa per volta può fare.

Abbiamo bisogno di due memorie perché contemporaneamente si possa operare sulla memoria dati e scriverci dentro e dalla altra parte iniziare a prelevare la prossima istruzione.

**Hazard sui dati** sono dovuti a come sono scritti i programmi, si verifica quando devo attendere il completamento di un’istruzione precedente perché mi servono i dati su cui sta operando l’istruzione precedente.

Si può risolvere parzialmente questo problema con il bypassing consiste nel prelevare un dato da un buffer interno all’unità di elaborazione invece di aspettare che arrivino ad uno dei registri visibili ad un programmatore o dalla memoria.

La cosa più fastidiosa dal punto di vista degli hazard sui dati sono gli hazard load-use anche con la tecnica del bypassing avrò sempre una bolla nella pipe.

L’Hazard load-use è originato dal fatto che il dato letto dalla memoria da un’istruzione load non è ancora disponibile quando servirebbe ad un’altra istruzione.

**Hazard sul controllo** si verifica quando in qualche maniera l’azione di controllo che deve essere presa (in pratica salto o non salto “l’istruzione di branch”) dipende dall’istruzione precedente quindi se non finisce l’istruzione precedente io so se saltare o non saltare.

L’istruzione di branch determina il flusso di controllo(cioè dove si fa a finire) quindi il fetch della prossima istruzione dipende dalla branch.

Per risolvere questo problema devo aggiungere dell’hardware che mi permetta di valutare la condizione prima che venga fatto nell’alu, questa operazione addizionale viene fatta nello stadio id (es. per un beq viene aggiunto un comparatore) ma comunque perdo un colpo.

Questa soluzione non è conveniente per le pipeline lunghe (perché non riesco a decidere nel secondo stadio l’esito del confronto si genera un rallentamento maggiore ), una seconda soluzione consiste nella

predizione dei salti.

L’approccio più semplice consiste nel predire che il salto non venga mai preso, se la predizione si rileva corretta, la pipeline procede al massimo della velocità e solo quando il salto doveva essere preso si verifica uno stallo. (Predizione statica)

Un'altra tecnica di predizione è detta predizione dinamica dei salti, predittori di hardware dinamici determinano la predizione per ciascun salto in base al comportamento precedente di quell’istruzione di salto.

L’implementazione della predizione dinamica dei salti consiste nel memorizzare la storia di ciascun salto, ricordando quando è stato preso o no, e nell’utilizzare il comportamento precedente per predire il futuro.

**Che cos’è la predizione di salto?**

In pipeline profonde e superscalari il problema dei salti e significativo, l’utilizzo della predizione statica causerebbe una perdita di prestazioni eccessiva con più hardware a disposizione è possibile predire i salti.

Una seconda tecnica consiste nel verificare se l’ultima volta che un’istruzione di salto è stata eseguita il salto sia stato preso questa tecnica è chiamata Predizione Dinamica (Dynamic Branch).

Un modo per implementare questa strategia consiste nell’utilizzare un buffer di predizione dei salti, detto anche tabella della storia dei salti. È una piccola memoria indicizzata che contiene un bit che indica se il salto era stato eseguito o meno nell’ultima esecuzione.

Se predico che il salto non venga preso continuo con l’istruzione successiva a quella di salto altrimenti se predico che il salto venga preso continuo con il fetch dell’istruzione di dove dovrei andare a finire.

Se mi accorgo che la predizione è sbagliata devo scaricare la pipe perché tutto quello che c’è dentro non è buono perché sono andato a prendere le istruzioni in un posto dove non dovevo prenderle.

La predizione a 1 bit ha un inconveniente anche se un salto venisse quasi sempre effettuato, la predizione risulterebbe sbagliata almeno due volte invece che una sola volta, quando il salto non viene effettuato.

Per rimediare a questo inconveniente spesso si utilizza uno schema a due 2 bit, in uno schema a 2 bit la predizione deve risultare sbagliata 2 volte di seguito prima di essere modificata.

**ECCEZIONI**

**Come funzionano le eccezioni?**

L’unità di controllo di un processore è la parte più difficile da far funzionare correttamente ed è la parte più difficile da rendere veloce. Il problema maggiore è rappresentato dall’eccezioni e dagli interrupt i quali alterano il normale flusso sequenziale di esecuzione delle istruzioni.

Eccezione è un evento non previsto che interrompe l’esecuzione di un programma. Inizialmente era utilizzata per segnalare condizioni di overflow aritmetico, lo stesso meccanismo fu utilizzato per garantire il corretto funzionamento della comunicazione dei dispositivi di I/O.

Interrupt è un’eccezione che ha origine all’esterno del processore. Alcune architetture utilizzano il termine interrupt per indicare tutti i tipi di eccezione

TRAP o SW/INT : sono particolari istruzioni che servono ad attivare dei meccanismi di gestione delle eccezioni/interruzioni da software: in pratica sto facendo una chiamata al sistema operativo tramite una system call. Quindi I/O è gestito dal sistema operativo.

GESTIONE DELLE ECCEZIONI NELL’ ARCHITETTURA MIPS

L’operazione fondamentale che il processore deve compiere quando si verifica un’eccezione consiste nel salvare l’indirizzo dell’istruzione che l’ha generata nel EPC (program counter delle eccezioni ) e trasferire il controllo ad un indirizzo specifico del sistema operativo.

Il sistema operativo intraprenderà quindi le azioni più opportune (ad esempio terminando il programma segnalando un errore); il S.O. potrà terminare il programma o riprenderne l’esecuzione utilizzando l’EPC per determinare il punto da cui ripartire.

Un metodo alternativo consiste nel nell’utilizzare gli interrupt vettorizzati, nei quali l’indirizzo a cui si deve trasferire il controllo viene determinato dalla causa dell’eccezione stessa. Serviranno però due registri:

* EPC: un registro a 32 bit utilizzato per memorizzare l’indirizzo dell’istruzione che ha generato l’eccezione
* Causa: un registro nel quale va memorizzata la causa dell’eccezione.

INTERRUPT IMPRECISI o ECCEZIONI IMPRECISE: sono interrupt/eccezioni che non sono associate realmente all’istruzione che le ha causate.

INTERRUPT IMPRECISI o ECCEZIONI PRECISE: sono interrupt/eccezioni che sono sempre associate all’istruzione che le ha causate.

**Interrupt e chi li genera?**

È un segnale che indica “il bisogno di attenzione” da parte di un periferica finalizzata ad una particolare richiesta di servizio.

**Interrupt hardware:**

**Interrupt software:**

**PROCESSORI STATICI E DINAMICI**

**Processori con parallelizzazione statica e dinamica?**

Esistono due modalità per realizzare processori a esecuzione parallela, la cui principale differenza consiste nel modo in cui viene suddiviso il lavoro tra il compilatore e l’hardware; le decisioni possono essere prese staticamente (cioè durante la compilazione) o dinamicamente (cioè durante l’esecuzione). Il primo approccio è detto parallelizzazione statica dell’esecuzione mentre il secondo approccio viene detto parallelizzazione dinamica dell’esecuzione.

Con la parallelizzazione statica abbiamo HW più semplice ma il compilatore deve conoscere la CPU, quindi in questo tipo di implementazione si utilizza il compilatore per formare i pacchetti di istruzioni e gestire gli Hazard. In questo tipo di processore le istruzioni che vengono lanciate in esecuzione in parallelo nello stesso ciclo di clock costituiscono un pacchetto di istruzioni, che si può intendere come un’istruzione ampia contenete più istruzioni al suo interno.

È utile considerare il pacchetto di istruzioni come un’istruzione singola nella quale vengono codificate diverse operazioni in campi predefiniti; questa considerazione ha prodotto il nome originale di questo approccio VLIW.

I processori dotati di parallelizzazione dinamica sono anche noti come processori superscalari.

Nei processori superscalari più semplici, le istruzioni vengono eseguite in ordine e il processore decide se nessuna, una o più istruzioni posso essere avviate all’esecuzione nello stesso ciclo.

Con la parallelizzazione dinamica è la CPU in HW ad esaminare le istruzioni per capire quali possono essere lanciate in parallelo.

Ovviamente per ottenere buone prestazioni con questo tipo di processore occorre che il compilatore riordini le istruzioni in modo da rimuovere le dipendenze per aumentare il numero di istruzioni che possono essere lanciate in parallelo.

Molti processori superscalari estendono la versione base dell’unità che gestisce la parallelizzazione (più HW) dell’esecuzione in modo da includere un riordinamento dinamico dell’istruzioni, viene utilizzato per evitare gli hazard e gli stalli.

In un processore di questo tipo la pipeline viene suddivisa in tre unità principali: un’unità che preleva l’istruzione e l’avvia all’esecuzione, un gruppo di più unità funzionali e un’unità di consegna.

La prima unità preleva le istruzioni, le decodifica e invia ciascuna di esse alla corrispondente unità funzionale per l’esecuzione.

Ciascuna unità funzionale è dotata di buffer, chiamati stazioni di prenotazione che conservano gli operandi e l’operazione.

Non appena la prenotazione contiene tutti gli operandi richiesti e l’unità funzionale è pronta per l’esecuzione, viene eseguita l’operazione e inviato il risultato alla stazione di prenotazione che la sta attendendo e contemporaneamente all’unità di consegna.

Questa conserva il risultato dell’operazione fino a quando non avrà il via libera per salvarlo nel register file o nel caso di un’istruzione di store nella memoria.

La differenza tra un processore superscalare e un processore VLIW è che nel primo è l’HW che garantisce che il codice riordinato o meno venga eseguito correttamente ed inoltre il codice in linguaggio macchina verrà sempre eseguito correttamente mentre in alcuni processori VLIW ciò può essere non vero e diventa necessario ricompilare i programmi quando ci sposta su modelli diversi dello stesso processore.

**Processore very long instraction word?**

Un processore VLIW (anche detto processore a parallelizzazione), in questo tipo di implementazione il compilatore viene utilizzato per formare i pacchetti di istruzioni e gestire gli hazard.

Le istruzioni che vengono lanciate nello stesso ciclo di clock (in esecuzione parallela) costituisco un pacchetto di istruzioni, che si può intendere come un’istruzione ampia contente più istruzioni al suo interno.

È utile considerare il pacchetto di istruzioni come un’istruzione singola nella quale vengono codificate diverse operazioni in campi predefiniti; questa considerazione ha prodotto il nome VLIW.

Quindi il VLIW è un tipo di architettura in grado di lanciare in esecuzione più istruzioni, indipendenti tra loro, attraverso un’istruzione codificata su molti bit la quale tipicamente contiene diversi codici operativi.

**Che cos'è il renaming dei registri?**

Il renaming dei regsitri è un processo compiuto dal compilatore o dall’HW per evitare le false dipendenze tra i dati che potrebbero generare hazard o limitare il riordino del codice.

False dipendenze si verificano quando abbiamo delle istruzioni completamente indipendenti tra di loro e tra di loro non esiste alcun flusso di dati ed è causata dall’utilizzo di uno stesso nome, tipicamente di un registro e non da una vera e propria dipendenza tra i dati delle istruzioni.

**Aggiunte Johnny**

La pipeline sfrutta il parallelismo potenziale esistente tra le istruzioni.

Ci sono due metodi principali per incrementare potenzialmente la quantità di parallelismo a livello di istruzioni. Il primo consiste nell’aumentare la profondità della pipeline per sovrapporre più istruzioni.

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 che si dà a questa tecnica è esecuzione parallela. (Funziona ni abbiamo il problema delle dipendenze)

Loop unrolling è una tecnica utilizzata per ottenere prestazioni migliori sui cicli.

In questa tecnica il corpo del ciclo viene replicato più volte e le istruzioni provenienti dalle varie repliche possono essere inserite negli stessi pacchetti di istruzioni. Riduce le dipendenze.

La pipeline è una tecnica di implementazione Hardware grazie alla quale l’esecuzione di più istruzioni viene sovrapposta nel tempo.

Le pipeline non diminuiscono il tempo di esecuzione ma il tempo totale per eseguire tutte le istruzioni (Throughput).

Se tutti gli stadi delle pipe richiedessero un tempo di esecuzione uguale e ci fosse abbastanza lavoro da fare allora l’aumento di velocità dovuto alle pipeline sarebbe pari al numero dei suoi stadi.

La pipeline introduce inoltre dei tempi che si aggiungono al tempo di esecuzione e in definitiva il tempo di esecuzione di una singola istruzione sarà sempre maggiore del minimo teorico, e quindi l’incremento di velocità sarà minore del numero degli stadi.

Le istruzioni in MIPS richiedono tipicamente 5 passi:

1. **IF**: fetch dell’istruzione; l’istruzione viene prelevata dalla memoria istruzioni.
2. **ID:** lettura dei registri e decodifica dell’istruzione (lettura e decodifica posso essere simultanee in MIPS il formato regolare dell’istruzioni lo permette).
3. **EX :** Esecuzione di un’operazione o calcolo dell’indirizzo.
4. **MEM:** accesso a un operando nella memoria dati.
5. **WB:** scrittura del risultato in un registro

STALLO: sospensione di una unità della pipeline (e delle precedenti)

STALLO: stato in cui si trova il del processore quando le istruzioni sono bloccate.