12. Ciclo Fetch-Execute, Parallelismo e Pipeline

Ciclo di Esecuzione delle Istruzioni

Il ciclo fetch/execute si articola in fasi distinte: calcolo dell'indirizzo dell'istruzione, fetch dell'istruzione, decodifica, calcolo dell'indirizzo degli operandi, fetch degli operandi, esecuzione dell'operazione, memorizzazione del risultato e controllo degli interrupt. Il flusso di dati coinvolge registri fondamentali: il Program Counter (PC) contiene l'indirizzo della prossima istruzione, il Memory Address Register (MAR) specifica l'indirizzo di memoria da accedere, il Memory Buffer Register (MBR) contiene i dati letti o da scrivere, e l'Instruction Register (IR) memorizza l'istruzione corrente.

Parallelismo

Il parallelismo consente di eseguire più attività contemporaneamente, riducendo il tempo totale di completamento. Con un singolo esecutore, tre lavori richiedono nove unità di tempo; con tre esecutori operanti in parallelo, gli stessi lavori terminano in tre unità di tempo.

Quando esiste dipendenza funzionale tra lavori successivi, ogni lavoro viene suddiviso in fasi sequenziali. Due approcci sono possibili: gli esecutori generici completano ciascuno un intero lavoro, richiedendo risorse replicate per ogni fase; gli esecutori specializzati svolgono sempre la stessa fase su lavori diversi, utilizzando meno risorse complessive. A regime, entrambi raggiungono lo stesso throughput del parallelismo totale.

Pipeline

La pipeline applica il principio della catena di montaggio all'esecuzione delle istruzioni. Un lavoro viene decomposto in fasi successive, ciascuna realizzata da un operatore diverso. In ogni istante, prodotti diversi si trovano in fasi diverse, realizzando parallelismo. Nell'istante successivo, ogni fase ripete lo stesso lavoro sul prodotto seguente e ogni lavoro avanza alla fase successiva.

Le fasi tipiche del ciclo esecutivo sono: fetch dell'istruzione (FI), decodifica (DI), calcolo dell'indirizzo degli operandi (CO), fetch degli operandi (FO), esecuzione (EI) e scrittura del risultato (WO). Ogni fase è realizzata da una diversa unità funzionale della CPU, e tra fasi successive si inseriscono buffer per i dati temporanei.

Performance della Pipeline

Il tempo di ciclo τ è determinato dalla fase più lenta più il ritardo di commutazione dei registri: $\tau = \max(\tau_i) + d$. Il tempo totale per eseguire n istruzioni con k stadi è $T_k = [k + (n-1)]\tau$. Lo speedup risulta $S_k = nk/[k + (n-1)]$, che si avvicina al numero di stadi k al crescere del numero di istruzioni. La pipeline non riduce il tempo di esecuzione di una singola istruzione, ma aumenta il throughput complessivo.

Pipeline Hazards

Quattro tipi di criticità impediscono il raggiungimento del parallelismo massimo.

Sbilanciamento delle Fasi

Le fasi hanno durate diverse e non tutte le istruzioni richiedono le stesse fasi. Il tempo di attesa forzato si verifica quando una fase deve attendere il completamento della precedente prima di passare i dati. Le soluzioni includono la decomposizione delle fasi onerose in sottofasi o la duplicazione degli esecutori delle fasi più lente.

Problemi Strutturali

Due o più fasi competono per la stessa risorsa nello stesso ciclo di clock, tipicamente l'accesso alla memoria da parte di FI, FO e WO. Le soluzioni prevedono l'introduzione di fasi non operative (nop) o la suddivisione delle memorie in cache separate per istruzioni e dati.

Dipendenza dai Dati

Un'istruzione necessita del risultato di un'istruzione precedente ancora in pipeline. I tipi di hazard sono: Read after Write (l'istruzione successiva legge prima che la precedente scriva), Write after Write (scrittura fuori ordine) e Write after Read (caso raro).

Le soluzioni comprendono: introduzione di stalli (nop), data forwarding (propagazione anticipata del risultato dall'uscita della ALU all'ingresso della fase che lo richiede, riducendo gli stalli) e riordino delle istruzioni da parte del compilatore per distanziare le dipendenze.

Dipendenza dal Controllo

Le istruzioni di salto, chiamate a procedura e interrupt alterano la sequenzialità. Quando si verifica un salto condizionato, le istruzioni già caricate nella pipeline dopo il branch potrebbero essere errate e vanno scartate, causando una penalità (branch penalty).

Soluzioni per i Salti Condizionati

Flussi multipli: replica la parte iniziale della pipeline per entrambi i rami possibili, con problemi di conflitti di risorse e complessità crescente per salti annidati.

Prefetch dell'istruzione target: carica anticipatamente sia l'istruzione sequenziale che quella target; una delle due verrà scartata.

Buffer circolare (loop buffer): una piccola memoria veloce mantiene le ultime n istruzioni prelevate, evitando il fetch quando il target è già presente. È particolarmente efficace per i cicli.

Predizione dei salti: approcci statici (saltare sempre, non saltare mai, decidere in base al codice operativo) e dinamici. La predizione dinamica a 1 bit ricorda l'esito dell'ultimo salto e predice lo stesso comportamento, commettendo due errori per ciclo a regime. La predizione a 2 bit richiede due errori consecutivi per invertire la predizione, riducendo gli errori a uno per ciclo. I bit di predizione sono memorizzati in una branch history table insieme all'indirizzo dell'istruzione di salto e all'indirizzo target.

Salto ritardato (delayed branch): l'istruzione nel branch delay slot viene sempre eseguita, indipendentemente dall'esito del salto. Il compilatore inserisce nel delay slot un'istruzione indipendente precedente al salto ("from before") oppure l'istruzione target ("from target", efficace quando il salto è probabile).

Esempio: Intel 80486

La pipeline dell'80486 comprende cinque stadi: Fetch (con buffer di prefetch da 16 byte), Decode 1 (decodifica opcode e modi di indirizzamento), Decode 2 (espansione in segnali di controllo e calcolo indirizzi complessi), Execute (operazioni ALU e accesso cache) e Writeback (aggiornamento registri e memoria). Il processore utilizza bypass per gli accessi consecutivi allo stesso dato e speculative fetch per le istruzioni target dei salti condizionati.