Architettura - approfondimento

Rappresentazione dei numeri secondo lo standard IEEE754

Complemento a 2: un numero intero N compreso tra 2^{n-1}-1 e -2^{n-1} si rappresenta su n bit effettuando la sottrazione 2^n-N e prendendo gli n bit meno significativi del risultato. Con 8 bit si possono rappresentare i numeri da 127 a -128, il primo bit rappresenta il segno. La somma da' un risultato corretto, ma se il risultato esce dal campo di rappresentabilità si genera un overflow. Per trasformare in complemento a 2 un numero basta convertirlo in complemento a 1 (invertire 0 e 1) e sommare 1.

I numeri in virgola mobile si rappresentano in forma $n = f * 10^e$, dove e è l'esponente, f è la parte decimale (*mantissa*). La precisione è determinata dal numero di cifre decimali, e la grandezza è determinata dal numero di cifre dell'esponente. La forma standard implica una parte decimale compresa tra 0 e 1, la forma normalizzata implica 1 sottointeso.

Forma *denormalizzata*, l'esponente è 00000000_2 e non ha sottinteso l'1. Questo permette di rappresentare numeri più piccoli del minor numero normalizzato. L'esponente quindi può essere 0, e la perdita di precisione è più graduale.

Ci sono 3 tipi di rappresentazione in virgola mobile (precisione singola e doppia sono in base 2 e hanno arrotondamento per eccesso):

- Precisione *singola* (32 bit), un bit per il segno, 8 per l'esponente e 23 per la mantissa;
- Precisione *doppia* (64 bit), un bit per il segno, 11 per l'esponente e 52 per la mantissa;
- Precisione *estesa* (80 bit, per ridurre al minimo gli errori di arrotondamento).

Come rappresentare un numero decimale in floating point a precisione singola:

1) Separare la parte decimale dalla parte intera, esempio 13.25 sarà 13 e 25;
2) Rappresentare la parte intera in binario (quindi 1101);
3) Moltiplicare la parte decimale per 2 finchè non si arriva a 100, se il 100 viene superato si mette 1, altrimenti 0 (quindi 25 è 01);
4) Unendo di nuovo le due parti si ottiene intero.decimale (1101.01);
5) Normalizzare, cioè spostare la virgola fino ad avere un solo bit 1 a sinistra (1.10101);
6) Moltiplicare il numero ottenuto per 2^n dove n è il numero di spostamenti della virgola;
7) Il primo bit rappresenta il segno, 1 negativo, 0 positivo, invece la parte decimale (10101) viene inserita nei 23 bit della mantissa a partire dal più significativo;
8) L'esponente è in eccesso 127, quindi per rappresentarlo in floating point bisogna aggiungere 127 alla n (esponente del 2) e convertirlo in binario, quindi 130 (somma di 127 e il 3 di 2^3) è 10000010_2;
9) I bit vuoti della mantissa sono 0, il risultato finale sarà 0 10000010 101010000000000.

Altre casistiche possibili a seconda del valore dell'esponente, oltre alla normalizzata:

- Zero, l'esponente è 00000000_2. Può essere scritto in due modi equivalenti, +0 e -0.

- Infinito/NaN (not a number, numero non valido), esponente 11111111_2. L'infinito ha 1 soltanto nell'esponente; NaN ha 1 anche nella mantissa.

Errore di approssimazione assoluto: la differenza tra il valore esatto e il valore approssimato.

Errore di approssimazione relativo: il rapporto tra l'errore assoluto e il valore approssimato. Si usa per capire la precisione dell'approssimazione; la precisione è definita dal numero di cifre della mantissa.

Operazioni in virgola mobile

Per tutte le operazioni si presuppone un formato di precisione singola (32 bit).

Somma di decimali:

1) Si presuppone che il valore assoluto del primo addendo sia maggiore dell'altro, in caso contrario scambiarli;
2) L'esponente del risultato è il maggiore tra i due esponenti, quindi il primo;
3) Si calcola la differenza tra i due esponenti, e si shifta a sinistra la virgola della seconda mantissa di tante posizioni quante il numero ottenuto dalla differenza (in questo modo entrambi gli esponenti sono uguali);
4) Sommare (o sottrarre, in base al segno) le mantisse ed eventualmente normalizzare il risultato aggiungendo 1 all'esponente;
5) Per la sottrazione in floating point basta invertire il segno del secondo addendo.

Prodotto di decimali:

1) Se uno degli operandi è 0 oppure infinito, si rappresenta il risultato con esponente 00000000 oppure 11111111 rispettivamente;
2) Si fa uno XOR tra il segno del primo operando e quello del secondo (se sono diversi, XOR darà come output 1, quindi segno negativo);
3) Le mantisse vengono moltiplicate (incluso l'1 sottinteso), e il risultato è la mantissa del prodotto arrotondata a 23 bit;
    a) NB: il risultato ha 48 bit, il prodotto si effettua come i prodotti decimali a più cifre in colonna, shiftando di una posizione ogni volta e infine sommando tutti i prodotti parziali;
    b) Se ci sono due cifre (di cui la più significativa è 1) prima della virgola, si aggiunge 1 all'esponente e si sposta nuovamente la virgola in modo da normalizzare il risultato;
4) Gli esponenti vengono sommati (es. $2^3 * 2^4 = 2^7$) ed è aggiunto l'eventuale 1 ottenuto dalla mantissa;
    a) NB: per ottenere 2^n è necessario togliere 127 (eccesso 127).

Divisione di decimali (assumendo che siano diversi da 0):

1) Il segno si trova con uno XOR dei segni;
2) La mantissa si ottiene dividendo le due mantisse, con una semplice divisione in colonna uguale a quella dei decimali;
3) L'esponente è ottenuto dalla sottrazione tra i due esponenti più 127;

4) Il risultato viene eventualmente normalizzato.

Per queste operazioni viene usata una ALU a 32 bit, con uno shift right/left.

Controllo del percorso dei dati in pipelining

Il *pipelining* è una tecnica in cui più istruzioni sono eseguite in parallelo (esempio della lavatrice: quando il primo carico è stato lavato, mentre si asciuga viene lavato il secondo carico, e così via). I componenti hardware, rispetto al datapath, sono gli stessi; ma a parità di numero di istruzioni la rapidità di esecuzione è maggiore. Tutti gli step (*stages*) operano contemporaneamente usando mezzi differenti; la velocità è uguale al numero di step.

Il tempo tra un'istruzione e l'altra è dato dal rapporto tra il tempo tra le istruzioni senza pipelining e il numero di stages. Questo risultato corrisponde anche al rapporto tra il tempo totale impiegato con datapath e quello con pipelining.

Le cinque fasi del pipelining (ognuna impiega un ciclo di clock) sono:
- Fetch, l'istruzione viene presa dalla memoria;
- Decode, lettura e decodifica dell'istruzione;
- Execute, esecuzione o calcolo dell'indirizzo;
- Accesso dell'operando nella memoria;
- Scrittura del risultato in un registro (write-back).

Le istruzioni prese in considerazione sono add, sub, lw, sw, and, or, slt e beq.

Ci sono tre casi (*hazards*) in cui l'istruzione successiva non può essere eseguita nel successivo ciclo di clock, che sono:
1) *Structural* hazards, quando l'hardware non supporta la combinazione di istruzioni e si verifica spesso con le operazioni in floating point (viene evitato aggiungendo componenti, come una memoria aggiuntiva);
2) *Data* hazards, quando il pipelining dev'essere interrotto (stallo) perchè uno step per essere eseguito deve aspettare che il precedente sia completato (esempio: add e sub, in cui la sub utilizza il risultato dell'add), per cui potrebbero essere necessari componenti hardware extra per recuperare i risultati (*forwarding*);
3) *Control* hazards, un problema nei programmi basati su integers, quando l'istruzione che è appena stata eseguita non è quella necessaria per l'istruzione corrente, si verifica uno stallo (l'istruzione che causa questo hazard è il branch, quindi il pipelining presuppone che la condizione non sia rispettata per cercare di non perdere tempo). Un modo avanzato di prevenire gli stalli (*prediction*) è anche la memorizzazione degli esiti dei branch in modo da poter prevedere comportamenti simili. MIPS utilizza le delayed decisions, cioè esegue prima l'istruzione dopo il branch, e poi il branch.

I programmi in floating point solitamente hanno una frequenza di branch più bassa, quindi è più semplice gestire e prevenire gli hazards.

Il datapath a pipelines a singolo ciclo implementa tutte le 5 fasi del processo, con le istruzioni e i dati che si spostano da sinistra verso destra tranne il write-back (che scrive il

risultato nel register file) e la scelta tra PC+4 e il branch address. Queste eccezioni nel flusso possono rispettivamente generare data hazards e control hazards.

Ogni fase è separata dai pipeline registers (IF/ID, ID/EX, EX/MEM, MEM/WB), che devono essere abbastanza ampi da contenere tutti i dati che ci passano attraverso.

Fasi nelle load instructions utilizzando il pipelining:

1) Fetch: l'istruzione viene letta utilizzando l'indirizzo nel PC, e viene inviata a IF/ID. Il PC viene sovrascritto aumentato di 4, e il nuovo indirizzo viene anch'esso inviato a IF/ID (che quindi viene scritto) per poterlo usare in caso di istruzioni come beq.
2) Decode: IF/ID viene letto, recuperando il campo a 16 bit nelle I-Type che viene esteso a 32, e l'indirizzo dei due registri. I tre valori sono salvati in ID/EX insieme al PC.
3) Execute: il contenuto dei registri in ID/EX viene letto e R1 è sommato al valore immediato utilizzando l'ALU. I risultati vengono scritti in EX/MEM.
4) Memory access: EX/MEM trasferisce i dati in data memory, che a sua volta li scrive in MEM/WB tramite read data.
5) Write-back: il contenuto di MEM/WB viene scritto nel register file (da destra verso sinistra) passando attraverso un multiplexer.

Fasi nelle store instructions utilizzando il pipelining:

1) Fetch: uguale alle load, PC+4 e lettura dell'istruzione, scrittura di IF/ID.
2) Decode: uguale. Gli stage di fetch e decode sono gli stessi per ogni istruzione.
3) Execute: il valore del secondo registro e i dati vengono scritti in EX/MEM.
4) Memory access: utilizzando il contenuto di EX/MEM, i dati sono trasferiti nella memoria (data memory).
5) Write-back: non succede niente, perchè la memoria è già stata scritta.

Ogni componente del datapath può essere utilizzata soltanto in un singolo stage, altrimenti si incontrano structural hazards. Per evitare bugs, è frequente che parti non immediatamente necessarie delle istruzioni siano trasferite da una pipeline all'altra in modo da non perdere indirizzi o dati. Tutte le pipeline sono collegate direttamente tra di loro, e anche collegate con componenti hardware in mezzo. MEM/WB è collegata al write register nella parte dei registri che si trova tra IF/ID e ID/EX.

I due tipi di base di rappresentazione del pipelining sono a ciclo di clock singolo e a ciclo di clock multiplo. I primi mostrano un datapath in cui i componenti vengono letti e/o scritti (si ricorda che tutte le operazioni vengono eseguite quasi contemporaneamente, quindi gran parte dei componenti sono evidenziati), i secondi rappresentano le 5 fasi orizzontalmente e le istruzioni verticalmente. Visto che le istruzioni non aspettano il completamento delle precedenti per essere lette ed eseguite, il numero di operazioni crescerà da 1 a 5 progressivamente ai cicli di clock.

Il controllo del pipelining avviene tramite control lines, similmente al datapath multiciclo. Nei primi 2 stage non c'è nessun controllo particolare dato che sono sempre uguali e non ci sono operazioni rilevanti che potrebbero causare errori.
Nella fase di execute sono settati RegDst, ALUOp e ALUSrc che selezionano il registro,

l'operazione dell'ALU e read data 2 oppure il valore immediato.
In memory access Branch, MemRead e MemWrite sono settati rispettivamente dalle istruzioni di beq, load e store.
Write-back ha i segnali MemtoReg e RegWrite che stabiliscono se i dati devono essere salvati nella memoria o in un registro.

Le dipendenze tra le istruzioni causano data hazards che sono risolti in diversi modi. Per esempio, se il valore di un registro cambia nel ciclo di clock 5, tutte le istruzioni successive sarebbero già state parzialmente eseguite usando il vecchio valore; oppure lo stesso registro potrebbe essere letto e scritto nello stesso ciclo di clock. Il secondo caso viene risolto dividendo ogni ciclo di clock in 2, lettura e scrittura; il primo caso viene risolto tramite forwarding. Ciò significa che il risultato, non appena è disponibile, viene inviato alle unità a cui servirebbe ancora prima di essere scritto in memoria.

Ci sono quattro tipi di dipendenze, in ognuna il register destination in EX/MEM o MEM/WB viene spostato in ID/EX come primo o secondo register source. Queste vengono gestite da due segnali di controllo a 2 bit, ForwardA e ForwardB. Per evitare forwarding non necessario, basta controllare se RegWrite è attivo e se RegisterRd è diverso da 0. La velocità aumenta grazie ai multiplexers che permettono il forwarding da ogni pipeline.

Un altro modo per evitare i data hazards è l'aggiunta di un'unità di rilevamento hazards, che mette in stallo la pipeline se ci sono incoerenze tra i registri (fa un if). Se si verifica uno stalling, nel frattempo le pipelines interessate eseguono istruzioni vuote dette nops, che significano letteralmente "non fare nulla". I control fields del registro proveniente da ID/EX sono settati a 0, quindi non avviene scrittura.

I control hazards avvengono quando l'hardware non sa il risultato di un branch e inizia a eseguire le istruzioni subito dopo. Una soluzione è annullare tutte le istruzioni in corso in caso di esito positivo del branch (flush), assumendo sempre l'esito negativo; oppure l'esito del branch viene valutato e trasmesso alle pipelines il prima possibile, riducendo il numero di istruzioni da scartare. Un altro modo è cercare di prevedere il risultato del branch tramite cronologia e tabelle, usando 2 bit in modo da aumentare la precisione.

La gestione delle eccezioni avviene similmente al datapath multiciclo: c'è un EPC dove viene scritto l'indirizzo dell'istruzione che ha generato l'eccezione, il registro Cause, e il controllo viene affidato al sistema operativo. Anche nel pipelining può essere usata la vettorizzazione delle interruzioni, dove si usa l'indirizzo del vettore per determinare la causa.

Nell'architettura a pipelining si usa il flushing, le istruzioni in corso vengono scartate e l'esecuzione riprende dal nuovo indirizzo nel PC. L'interruzione può verificarsi in ID/EX o IF/ID, quindi da essi partono segnali di controllo per il flushing. Infine PC-4 viene salvato in EPC. Nel caso in cui più eccezioni siano rilevate nello stesso ciclo di clock, la priorità solitamente è a quella nella prima istruzione.

La moltiplicazione in pipelining può essere effettuata ancora più velocemente per merito dei carry save adders.

- Controllo del percorso dei dati in pipelining;
    - 236-248
    - Dopo pagina 248, non credo sia necessario
    - Slide ed esercizi Moodle
- Cache direct, n-way set associative, fully associative;
    - 347-357
- Sviluppo da zero di piccoli programmi in assembly MIPS32.
    - Allenarsi a scrivere programmi
    - Convenzioni
    - Rivedere bene gli *handler*
- Gestione delle priorità nelle interruzioni.
    - Slide Moodle
    - ?????????

In generale:

http://www.cs.umd.edu/class/spring2003/cmsc311/Notes/index.html