Appunti Architettura

Brendon Mendicino

November 6, 2022

Contents

1 Introduzione **3**

2 Pipeline **3**

3 Pipeline Hazards **4**

4 Floating Point **4**

5 Exceptions **5**

6 Chache Memories **6**

7 Branch Prediction **9**
7.1 BHT 9

8 Pipelining **10**
8.1 Static Scheduling 10
8.2 Dynamic Scheduling 10

9 ARM **12**

1 Introduzione

2 Pipeline

Per misurare le prestazioni di una pipeline si usa il **throughput**. Il throughput è il numero di sitruzioni che escono dalla pipeline in un intervallo di tempo.

Il datapath è composto da:

- instruction fetch: si prende dalla memoria la prossima istruzione putnatata dal PC e si incrementa quest'ultimo di 4;
- decode: si decodifica l'istruzione, attivando il datapath in modo adeguato, a prescindere dal tipo di operazione carico i due registri rs ed rt, ed il campo immediato, anche nel caso l'istruzione non fosse immediata, risparmio del tempo aumentando leggermente il comsumo di potenza;
  - A ¡- Reg[rs];
  - B ¡- Reg[rt];
  - Imm ¡- ($IR_{16...31}$);
- execution/effective address cycle:
  - memory reference: ALUoutput ¡- A + Imm;
  - register-register: ALUoutput ¡- A op B;
  - register-immediate: ALUoutput ¡- A op Imm;
  - brach: ALUoutput ¡- NPC + Imm; Cond ¡- (A op 0);
- memory access/branch completion cycle:
  - LMD ¡- Mem[ALUoutput]; or Mem[ALUoutput] ¡- B;
  - branch: if (cond) PC ¡- ALUoutput else PC ¡- NPC;
- write-back cycle:
  - register-register: Reg[$IR_{16...20}$] ¡- ALUoutput;
  - ...

L'assunzione molto forte sarà che tutti i dati e le istruzioni saranno sempre nelle memoria cache, quindi si avrà un delay di un solo colpo di clock. Il registre file potrà sia essere letto che scritto, ci sarà dunque bisogno di soddisfare queste richeste in un solo colpo di clock, la scrittura avviene nella prima parte del colpo di clock mentre la lettura avviene nelle seconda parte del colpo di clock.

Si aggiungono dei registri (detti pipeline register),

Aggiungere i registri della pipeline aggiunge un overhead, inoltre il clock del processore comporta un rallentamento, causato dallo skew.

3 Pipeline Hazards

Sono dei casi in cui l'istruzione non viene esguita in modo corretto:

- structural hazards: dipende dalla memoria,
- data hazards: dipende da come i dati vengono scritti e letti;
- control hazards: dependi dai branch;

Stall Un modo di gestire gli hazards è di manadre la CPU in stallo.

Risolvere gli hazard strutturali comporta un costo, in termini di nuovo hardware e di migliorare quello esistente. Un processore con hazard strutturali avrà un clock più veloce ma problemi di accesso alla memoria, un processore senza hazard strutturali avrà un clock più lento ma nessuna limitizione di accesso alla memoria.

Data hazards Generati dalle dipendenze dei dati generati all'interno della pipeline. Esempio:

```
1 add r1, r2, r3
2 sub r4, r1, r5
3 and r6, r1, r7
4 or  r8, r1, r9
5 xor r10, r1, r11
```

Il registro r1 viene inizializzato nella prima istruzione e poi utilizzato nel resto del codice, ma l'operazione di scrittura in si trova alla fine, infatti l'istruzione successiva (sub) dovrebbe aspettare che r1 sia scritto prima che possa essere letto, se tuttavia proviamo a leggere r1 il risultato sarà non deterministico.

Per risolvere questo problema si hanno due soluzioni:

- mandare in stallo il processore;
- implementiamo un forwarding che permette di non attendere la scrittura del registro, ma di leggere direttamente il valore dei registri della pipeline;

4 Floating Point

Le operazioni floating point sono molto complesse, se si cerca di implementarle in un solo colpo di clock allora il processore diventa troppo complicato dal punto di vista logico, oppure un altra suluzione potrebbe essere quello di rallentare il clock, per far entrare tutte le operazioni in un singolo colpo, ma entrambe le soluzioni non sono fattibili, allora si prende un approcia di suddivisione della pipeline in unità. Per supportare le operazioni di floating point in pipeline, si è optato per una separazione dalla execute in diverse unità:

- integer unit;
- fp/integer multiply;
- fp adder;
- fp/integer divider;

Questa ramificazione della pipeline va a convergere nella sezione di MEM.

Si dovrà definire la latenza, ovvero il numero di colpi di clock che una unità usa per avere un risultato, ed un intervallo di inizializzazione, ovvero il numero di colpi di clock che la seconda istruzione davrà attendere per entrare nella sezione desiderata (come somma o divisione). Un esempio:

- add: lat: 1, int: 1;
- mult: lat: 8, int: 1;
- fadd: lat: 4: 1;
- div: lat: 24, int: 24;

Solitamente la divisione ha la latenza identica all'intervallo di inizializzazione. Solitamente su un unità è **pipelined** allora ha un colpo di clock come intervallo di inizializzazione, se l'unità non è pipelined allora il suo intervallo di inizializzazione è uguale alla latenza.

Un altro porblema dato dagli hazard strutturali è il fatto che: più istruzioni non possono accedere in contemporanea alla fase di MEM o di WB, solitamente il criterio è FIFO, oppure si potrebbe dare maggiore priorità alla istruzioni con il maggiore numero di clock.

5 Exceptions

Le eccezioni sono classificate in:

- **sincrone** e **asincrone**;
- **user requested** (l'utente potrebbe creare un eccezione) o **coerced** (data da fattori esterni);
- **maskable** o **non-maskable**: alcune eccezioni non sono mascherabili ovvero forzare l'hardaware a non rispondere all'exception;
- **within instructions** o **between instructions**: within all'interno delle istruzioni (metre un istruzione fa DI, ID, ...) oppure tra due istruzinoi (ld, add);

Le macchine moderne sono chiamate **restartable machines** ovvero fanno ripartire il processore dallo stato in cui si trovava prima dell'eccezione. Quando una exception arriva il processore deve stoppare il IF, fermare la scrittura della pipeline e riuscire a tornare dalla procedura che ha chiamato l'exception.

Example 5.1 – Interrupt protocal in 80x86

Quando la CPU rileva un interrupt, legge il tipo di interrupt dal bus, salva lo stato del processore

ARM: la CPU salva lo status, il PC ed il Processore Status Register nello stack, aggiorna i flag e salta al valore dell'exception

Le eccezioni possono gestite in modo **preciso** o in modo **impreciso**:

- preciso: quando avviene un interruzione, tutte le istruzione prima che arrivi l'istruzione devono essere completate, tutte quelle dopo devono essere rimandate, gestire questo tipo di istruzioni è molto oneroso;
- impreciso:

Nel MIPS le possibili exception sono:

- IF: page fault, accesso a memoria protetta;
- ID: opcode illegale;
- EX: exception aritmetiche;
- MEM: stesse della IF;
- WB: nussuna;

Se due exception arrivano nello stesso momento: si può creare un flag di status associato ad ogni istruzione, guardande se l'istruzione può causare un eccezione, quando l'istruzione termina, l'exception viene scatenata, così si crea un coda evitando exception simultanee.

6 Chache Memories

Le cache volocizzano l'accesso al memoria secondaria che è il collo di bottiglia dell'intero sistema. Si è creata un gerarchia di memorie molto più veloci quanto più vicine si trovano al processore.

- reigistri: 500 bytes, 500ps;
- L1: 64 KB, 2ns;

- L2: 256 KB, 10-20ns;
- Memoria primaria: 512 MB, 50-100ns;
- Memoria secondaria flash: 8 GB, 50μs;

La cache funzionano grazie ai principi di località:

- temporale: in un tempo $t_0 + \Delta t$ dal momento in cui ho letto un elemento, è probabile che il dato venga o l'istruzioe venga riusato;
- spaziale: in un spazio $x + \Delta x$ vicino all'elemento letto, è probabile che gli elementi vicini vengano letti;

Theorem 6.1 – Cache Performance

- h: cache ratio;
- C: cache access time;
- M: memory access time;

Media del tempo di accesso:

$$t_{ave} = h * C + (1 - h) * M$$

Valori soliti per h sono 0.9.

Organizzazione della cache Solitamente la cache è formata da una parte di contrllo contenente il **cache controller** che controlla se accedendo alla cache è stato fatto un hit o un miss e in caso recupera la porzione di memoria ed una parte di dati, che contiene le **cache line** fatte da:

- validity bit: il bit ci dice se la riga è valida o meno;
- tag: identifica il blocco di memoria presente nella riga;
- data array: contiene i dati presi dalla memoria;

A partire dall'indirizzo la cache si calcola un nuovo indirizzo di accesso alle righe, foramto da:

- tag: identifica il blocco di memoria (bit dell'indirizzo - bit index - bit offset);
- index: riga della cache;
- offset: byte offset all'interno della riga;

Per regolare l'accesso alla cache il controllore identifica la riga attraverso l'index e comparando i due tag decide se è un hit o un miss, se è un hit ed il validity bit è a 1, allora attraverso l'offset viene prelevato il dato.

La cache si trovano tra il bus ed il processore, per evitare conflitti di utilizzo con perfiriferiche esterne o DMA.

Le cache moderne più vicine al processore sono separtate in **Instruction-Cache** e **Data-cache**, per evitare la lettura contemporanea di istruzioni e dati (structural hazard).

Mappatura I tipi di mappatura (**associativity models**) sono:

- **direct mapped**: la posizione nella riga è uguale a: #block_memory mod #cache_block;
- **set associative**: la cache viene partizionata in set di righe (i blocchi hanno tutti la stessa lunghezza, tipicamente 2 o 4), la posizione è determinata da: #memory_block mod #sets, quando un altro blocco viene asseganto ad un set viene rimossa la riga meno utilizzata;
- **fully associative**: ogni blocco di memoria può essere salvata in qualsiasi riga della cache, questo ha come malus la perdita del compo index e l'indirizzo del blocco viene salvato per intero;

Algoritmi di rimpiazzamento Gli aloritmi usati per decidere quale riga rimpiazzare sono:

- LRU (last recently used): il più usato;
- FIFO: il meno caro in termini di prestazioni;
- LFU (least frequently used): teorico, il più efficace;
- random: semplice ed efficace;

Update della Memoria Quando un operazione di scrittura è fatta sulla cache deve anche essere propagata sulla memoria, le due possibili soluzioni a questo problema sono:

- **write back**: per ongi riga della cache è introdotto un flag detto **dirty bit**, che indica quando i dati all'interno sono cambiati;
- **write through**: ogni volta che la CPU effettua un operazione di scrittura, i dati vengono scritti sia in cach e che in memoria;

7 Branch Prediction

Per effettuare delle predizioni esistono due tipi di approcci: prediction statici e prediction dinamici. Un esempio prediction statico è prendere tutti i salti come presi, oppure facciamo un filtro su quali tipi di branch prendere come presi (prendere i salti all'indietro come sempre presi).

Per avere delle predizioni più accurate è usare un brach prediction dinamico (speculazione). I metodi di predizione sono:

- branch history table;

Branch History Table Il BHT ha una memoria in cui sono contenute le informazioni relative ai salti. Si accede a questa tabella quando nella fase di fetch si prende un salto, si guarda l'informazione relativa e al salto e si legge la predizione del salto.

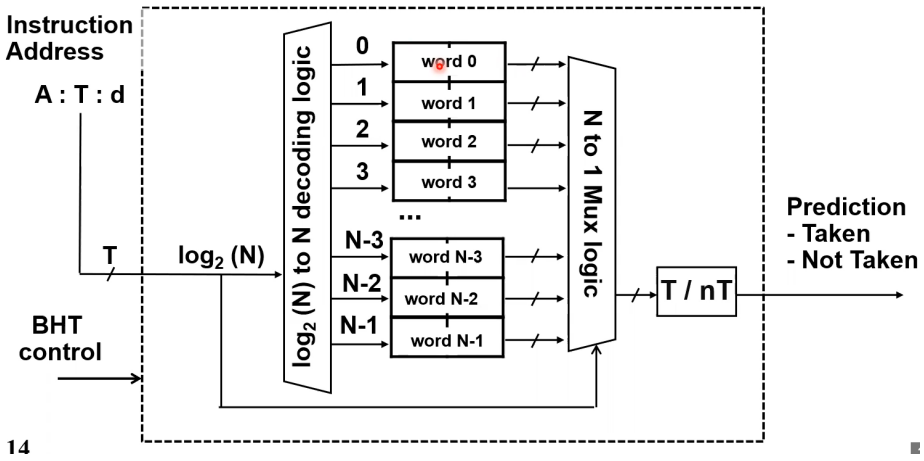

Figure 1: Bht Implementation

7.1 BHT

Un tipo di predittore dinamico: è una piccola memoria che salva per ogni istruzione salva dei valori funzionamento:

- se un branch non viene effettuta il contatore misP. counter ed il BHT vengono incrementati, il miss viene incrementato fino ad un massimo di differenza con il BHT, superato questo threshold i contatori non vengono incrementati ed il branch prediction è attivato per quella istruzione;
- se un branch viene effettuato il BHT viene decrementatto, se il BHT è minore del miss il branch prediction non viene attivato per quella istruzione;

...

8 Pipelining

Esistono delle tecniche per portare le CPI sotto il numero 1, questo è possibile farlo attraverso il fetch di più istruzioni. Esistono due tipi di processori che possono farlo:

- **superscalari**: hanno uno scheduling statico o dinamico;
- **very long instruction word (VLIW)**;

8.1 Static Scheduling

Per implementare un processore superscalare viene creato un **issue packet**, dove viene fatto il fetch di due (o più) istruzioni contemporaneamente se in modo statico: una è load, store, branch o operazioni ALU e l'altra è una qualsiasi operazione FP, queste due istruzioni vengono chimate issue packet.

Instruction type	Pipe stages							
Integer instruction	IF	ID	EX	MEM	WB			
FP instruction	IF	ID	EX	EX	EX	WB		
Integer instruction		IF	ID	EX	MEM	WB		
FP instruction		IF	ID	EX	EX	EX	WB	
Integer instruction			IF	ID	EX	MEM	WB	
FP instruction			IF	ID	EX	EX	EX	WB
Integer instruction				IF	ID	EX	MEM	WB
FP instruction				IF	ID	EX	EX	EX

Figure 2: Issue Packet Example

In un caso ideale si eseguiranno 0.5 istruzioni per colpo di clock. L'issue packet conterrà sempre una sola istruzione di branch. In questo caso l'unità FP sarà pipelined o indipendente, in qualche modo è possibile ottere degli hazard, come: fare un un istruzione di load e subito dopo un'istruzione di write, oppure dei possibili RAW (read after write).

Nei sistemi moderni si utilizza una strategia statica in alcuni processori embedded con MIPS.

8.2 Dynamic Scheduling

Si può ottenere una schedulazione dinamica Si ha un Common Data Bus (sistema di forwarding) comune, quindi viene duplicato. Suppoiamo di avere le seguenti istruzioni:

```
loop:
    ls r2, 0(r1)
    daddiu r2, r2, 1
    sd r2, 0(r1)
    daddiu r1, r1, 4
    bne r2, r3, loop
```

Supponiamo che:

- non ci sia speculazione:

Iteration number	Instructions		Issues at clock cycle number	Executes at clock cycle number	Memory access at clock cycle number	Write CDB at clock cycle number	Comment
1	LD	R2,0(R1)	1	2	3	4	First issue
1	DADDIU	R2,R2,#1	1	5		6	Wait for LW
1	SD	R2,0(R1)	2	3	7		Wait for DADDIU
1	DADDIU	R1,R1,#4	2	3		4	Execute directly
1	BNE	R2,R3,LOOP	3	7			Wait for DADDIU
2	LD	R2,0(R1)	4	8	9	10	Wait for BNE
2	DADDIU	R2,R2,#1	4	11		12	Wait for LW
2	SD	R2,0(R1)	5	9	13		Wait for DADDIU
2	DADDIU	R1,R1,#4	5	8		9	Wait for BNE
2	BNE	R2,R3,LOOP	6	13			Wait for DADDIU
3	LD	R2,0(R1)	7	14	15	16	Wait for BNE
3	DADDIU	R2,R2,#1	7	17		18	Wait for LW
3	SD	R2,0(R1)	8	15	19		Wait for DADDIU
3	DADDIU	R1,R1,#4	8	14		15	Wait for BNE
3	BNZ	R2,R3,LOOP	9	19			Wait for DADDIU

Figure 3: Dynamic Scheduling Senza Speculazione

- con speculazione:

Iteration number	Instructions		Issues at clock number	Executes at clock number	Read access at clock number	Write CDB at clock number	Commits at clock number	Comment
1	LD	R2,0(R1)	1	2	3	4	5	First issue
1	DADDIU	R2,R2,#1	1	5		6	7	Wait for LW
1	SD	R2,0(R1)	2	3			7	Wait for DADDIU
1	DADDIU	R1,R1,#4	2	3		4	8	Commit in order
1	BNE	R2,R3,LOOP	3	7			8	Wait for DADDIU
2	LD	R2,0(R1)	4	5	6	7	9	No execute delay
2	DADDIU	R2,R2,#1	4	8		9	10	Wait for LW
2	SD	R2,0(R1)	5	6			10	Wait for DADDIU
2	DADDIU	R1,R1,#4	5	6		7	11	Commit in order
2	BNE	R2,R3,LOOP	6	10			11	Wait for DADDIU
3	LD	R2,0(R1)	7	8	9	10	12	Earliest possible
3	DADDIU	R2,R2,#1	7	11		12	13	Wait for LW
3	SD	R2,0(R1)	8	9			13	Wait for DADDIU
3	DADDIU	R1,R1,#4	8	9		10	14	Executes earlier
3	BNE	R2,R3,LOOP	9	13			14	Wait for DADDIU

Figure 4: Dynamic Scheduling Con Speculazione

9 ARM