

# Appunti Architettura

Brendon Mendicino

October 25, 2022

---

## Contents

<b>1</b>	<b>Introduzione</b>	<b>3</b>
<b>2</b>	<b>Pipeline</b>	<b>3</b>
<b>3</b>	<b>Pipeline Hazards</b>	<b>4</b>
<b>4</b>	<b>Floating Point</b>	<b>4</b>
<b>5</b>	<b>Exceptions</b>	<b>5</b>
<b>6</b>	<b>Cache Memories</b>	<b>6</b>
<b>7</b>	<b>BPU</b>	<b>9</b>

# 1 Introduzione

## 2 Pipeline

Per misurare le prestazioni di una pipeline si usa il **throughput**. Il throughput è il numero di istruzioni 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 consumo di potenza;
  - $A \leftarrow \text{Reg}[\text{rs}]$ ;
  - $B \leftarrow \text{Reg}[\text{rt}]$ ;
  - $\text{Imm} \leftarrow (IR_{16...31})$ ;
- execution/effective address cycle:
  - memory reference:  $\text{ALUoutput} \leftarrow A + \text{Imm}$ ;
  - register-register:  $\text{ALUoutput} \leftarrow A \text{ op } B$ ;
  - register-immediate:  $\text{ALUoutput} \leftarrow A \text{ op } \text{Imm}$ ;
  - brach:  $\text{ALUoutput} \leftarrow \text{NPC} + \text{Imm}$ ;  $\text{Cond} \leftarrow (A \text{ op } 0)$ ;
- memory access/branch completion cycle:
  - $\text{LMD} \leftarrow \text{Mem}[\text{ALUoutput}]$ ; or  $\text{Mem}[\text{ALUoutput}] \leftarrow B$ ;
  - branch: if (cond)  $\text{PC} \leftarrow \text{ALUoutput}$  else  $\text{PC} \leftarrow \text{NPC}$ ;
- write-back cycle:
  - register-register:  $\text{Reg}[IR_{16...20}] \leftarrow \text{ALUoutput}$ ;
  - ...

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 richieste in un solo colpo di clock, la scrittura avviene nella prima parte del colpo di clock mentre la lettura avviene nella 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 eseguita in modo corretto:

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

**Stall** Un modo di gestire gli hazards è di mandare 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 limitazione 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 soluzione 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 approccio 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 dovrà 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, int: 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 problema 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 alle 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'hardware a non rispondere all'exception;
- **within instructions** o **between instructions**: within all'interno delle istruzioni (metà un'istruzione fa DI, ID, ...) oppure tra due istruzioni (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 protocol 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 istruzioni 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: nessuna;

Se due exception arrivano nello stesso momento: si può creare un flag di status associato ad ogni istruzione, guardando 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 velocizzano l'accesso alla memoria secondaria che è il collo di bottiglia dell'intero sistema. Si è creata una gerarchia di memorie molto più veloci quanto più vicine si trovano al processore.

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

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

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'istruzione 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 controllo 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, formato 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 periferiche esterne o DMA.

Le cache moderne più vicine al processore sono separate 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:  $\#block\_memory \bmod \#cache\_block$ ;
- **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:  $\#memory\_block \bmod \#sets$ , quando un altro blocco viene assegnato 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 campo index e l'indirizzo del blocco viene salvato per intero;

**Algoritmi di rimpiazzamento** Gli algoritmi 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 ogni 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 cache che in memoria;



## 7 BPU

...