Pipelining

Obiettivi

L'obiettivo del seminario è l'analisi dell'architettura interna dei processori che implementano modelli per la sovrapposizione nell'elaborazione di istruzioni ("parallelismo a livello di istruzioni") mediante l'impiego di pipelining e/o sulla duplicazione di unità funzionali. Vengono inoltre presentate le principali problematiche da affrontare nel progetto di sistemi di calcolo basati su tale classe di processori.

Le Tematiche Trattate

- Architetture basate su pipelining e/o sulla duplicazione funzionale (RISC, CISC, Superscalari, DSP, VILW, Vettoriali)
- Limiti nell'impiego del parallelismo
- Analisi dei processori commerciali (PowerPC, Pentium, Alpha)
- L'interazione tra l'architettura e il sistema operativo
- Progetto di sistemi distribuiti mediante nodi ad elevate prestazioni

Modello di riferimento (I)

Architetture di load/store : tutte le istruzioni che prevedono accesso alla memoria sono di tipo load o store e tutte le istruzioni che prevedono operazioni di tipo logico o aritmetico hanno operandi di tipo registro (Alpha, PowerPC). I processori possono differire per il modo di indirizzamento ammesso per i codici di load e store.

Modello di riferimento (II)
- Le Fasi del Processore DLX-

IF Prelievo Istruzione

ID Decodifica / Preparazione degli operandi da registri interni

EX Esecuzione / Calcolo dell'indirizzo effettivo

MEM Accesso alla memoria /Completamento dei salti

WB Scrittura in memoria.

Modello di riferimento (III)
- La Pipeline -

Istruzione i	*IF*	*ID*	*EX*	*MEM*	*WB*		
istruzione i+1		*IF*	*ID*	*EX*	*MEM*	*WB*	
istruzione i+2			*IF*	*ID*	*EX*	*MEM*	*WB*

Modello di riferimento (IV)

Il sistema è dotato di almeno una memoria di tipo cache per contenere dati e istruzioni (cache unica o cache separata per dati e istruzioni).

Modello esteso prevede lo studio di architetture dotate di più pipe composte di un numero diverso di stadi (in order o out order completion).

Pipeline Speed up

- Teoricamente, se gli stage della pipe sono perfettamente bilanciati e non si verificano condizioni di stallo, il tempo per ogni singola istruzione è:

$$T_i = \frac{\text{tempo per istruzione senza pipe}}{N_{stage} \text{ della pipe}}$$

- Il tempo per avere la prima istruzione è detto tempo di latenza (si ha ogni qual volta si verifica uno stallo)
- In realtà bisogna considerare il tempo di setup dei latch tra i vari stadi della pipe ed il bus skew

Calcolo dello SpeedUp

- Vedi appunti

Alcune ipotesi di funzionamento

1. La ALU è dotato di un ulteriore addizionatore per incrementare PC,
2. Ad ogni colpo di clock viene caricata una nuova istruzione (IF),
3. Ad ogni colpo di clock viene caricata una nuova word di dati (MEM),
4. Ci sono due registri separati per il LOAD e lo STORE dei dati (LMDR, SMDR),
5. L'accesso in memoria deve essere 2Nvolte più veloce rispetto alle macchine senza pipe, la cache è fondamentale e deve consentire l'accesso simultaneo alle due aree (dati ed istruzioni).

Limiti all'impiego della Pipeline

- Conflitti nell'utilizzo della pipe (Structural Hazards): non e' possibile sovrapporre l'esecuzione di tutte le istruzioni.
- Conflitti nell'utilizzo dei registri (Data Hazards): tali conflitti possono essere anche essere dovuti al diverso tempo di accesso alla memoria nel caso di dato non presente in cache.
- Variazioni nel flusso di caricamento delle istruzioni (Control Hazards), ad esempio salti.
- Gestione delle interruzioni (Imprecise / Precise Interrupts)

Conflitti nell'utilizzo della pipe

Il parallelismo interno alle istruzioni richiede la duplicazione di alcune risorse del sistema per consentire tutte le possibili combinazioni di istruzioni nella pipe.

In caso di structural hazard, la pipe va in stallo per uno o più cicli di clock (ad esempio se si ha una cache unica per dati ed istruzioni)

Load con un memory port

Istruzione i	*IF*	*ID*	*EX*	*MEM*	*WB*		
istruzione i+1		*IF*	*ID*	*EX*	*MEM*	*WB*	
istruzione i+2			*IF*	*ID*	*EX*	*MEM*	*WB*
istruzione i+3				***stall***	*IF*	*ID*	*EX*
istruzione i+4						*IF*	*ID*

Calcolo dello SpeedUp in caso di structural hazard

- Vedi appunti

Conflitti nell'utilizzo dei registri (I) Data Hazard

Tutti i conflitti, se non gestiti opportunamente, possono portare ad uno stato del processore operante in pipelining diverso da quello di un processore che non prevede tecniche di sovrapposizione nell'elaborazione di diverse istruzioni.

Ad ogni ciclo della pipe non viene rispettato il modello di Von Neumann tuttavia alla fine dell'elaborazione lo stato raggiunto è (deve) essere coerente con il modello.

Data Hazard (1)

ADD R1,R2,R3 ; R1 ← R2 + R3
SUB R4,R1,R5 R4 ← R1 – R5

IF	ID	EX	MEM	WB (dato scritto)	
	IF	ID (dato letto)	EX	MEM	WB

Data Hazard (2)

Una tecnica utilizzata per evitare un Data Hazard è il **Forwarding (o bypassing o ShortCircuiting),** consiste nel modificare l'architettura prevedendo che tutte le uscite della ALU siano retroazionate con dei registri latch e dei multiplexer agli Ingressi;

Se il forwarding Hardware si accorge che c'è un data hazard seleziona i Mux in modo che nella ALU vengano caricati i valori retroazionati e non il contenuto dei registri in conflitto (Figura 6.9).

Dunque, è possibile utilizzare subito il dato modificato nel registro R1 inserendo nell'architettura: multiplexer, buffer e un comparatore

Data Hazard (3)

- In caso di Data Hazard, il forwarding deve essere inoltrato anche alle 2 istruzioni successive (cioè a tutte le istruzioni che seguono fino a quando la fase di WB non è stata completata).
- Vedi Figure 6.7

Data Hazard (4)

- Per limitare il numero di istruzioni da bypassare, dato che ogni livello richiede dell'hw speciale (un buffer per le uscite della ALU per ogni istruzione bypassata), si usa il seguente accorgimento:
  - Dato che il banco registri viene acceduto due volte dalla pipe (ID e WB) si fa in modo che esso vengo scritto nella seconda fase di WB e letto nella prima fase di ID così nella i+3-ma istruzione le fasi di ID e WB possono essere contemporanee.
  - Figura 6.8

Data Hazard (5)

- I Data Hazard si possono avere anche quando due istruzioni cercano di accedere alla stessa locazione di memoria oppure quando ciè un cache miss in una istruzione e le successive proseguono …… nel DLX si manda in stallo tutta la pipe, altri processori permettono che il LOAD e STOre avvenga in un ordine differente da quello del programma con tecniche avanzate.

Data Hazard (6)

- Le tecniche di forwarding possono essere utilizzate non solo per la ALU ma anche verso altre Unità funzionali per evitare lo stallo; ex:
  - ADD R1,R2,R3
  - SW 25(R1),R1

Tipologie di Data Hazard

- **Read after Write** (come nel precedente esempio)
- **Write after Read** (non si può verificare in DLX dove in ID si legge e si scrive solo in WB)
- **Write after Write** (si verificano in architetture in cui si può scrivere in più stages della pipe e non solo in WB oppure in arch. In cui istruzioni possono procedere anche se ce ne sono alcune in stallo)
- **Read after Read** (non produce alcun conflitto)

Considerazioni

Alcune situazioni di stallo non sono possibili se la pipe è unica, le operazioni coinvolgono solo registri e le operazioni di scrittura possono avvenire solo in uno stadio della pipe.

Nel caso in cui le operazioni non coinvolgono solo registri si deve in generale avere prevedere uno "stallo" nell'utilizzo della pipe.

Data Hazard – pipeline interlock

LW R1, 32(R6)

ADD R4, R1, R7

Non basterebbe il forwarding dall'MBR

IF	ID	EX	MEM	WB		
	IF	ID	Stall	EX	MEM	WB

La tecnica che serve per mandare in stallo la pipe nel mezzo dell'istruzione e poi forwardare i dati appena arrivano dalla memoria si chiama pipeline interlock

Data Hazard – pipeline interlock

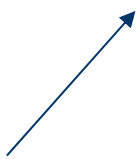
- Quando un data hazard deriva dall'attesa di un dato dalla memoria si preferisce mandare in stallo la pipe nel mezzo delle istruzioni e poi i dati vengono forwardati direttamente nella ALU,
- Per implementare questa tecnica occorrono dei comparatori che prendono in ingresso l'istruzione di load e la sua successiva per verificare il conflitto ed occorrono dei multiplexer e delle connessioni dirette tra unità differenti; la logica di controllo può essere implementata sia in Hw che direttamente dalla Control Unit del Processore.

Calcolo dello SpeedUp in caso di data hazard

Supponiamo che le operaz. Di LOAD sono circa il 20% e nel 50% dei casi dopo un LOAD c'è un operazione che dipende dal dato caricato.

Si supponga $CPI_{ideale} = 1$

$CPI = 0.8 * 1 + 0.2 * 1{,}5 = 1.1$



Durata di una operazione dopo il LOAD a causa dello stallo

Considerazioni

- Posso evitare di mandare in stallo la macchina modificando la sequenza delle istruzioni del programma;
- Ad esempio potrei mettere due load consecutivi e posticipare le operazioni che attendono le risorse dal primo load…..
- Questa tecnica prende il nome di **Pipeline scheduling**

Impiego di tecniche di schedulazione della Pipe

Tecniche implementative possono essere demandate:

- in parte al compilatore (tecniche statiche)
- essere gestite direttamente dal processore (tecniche dinamiche).

Ibernazione di una istruzione: Internal Forwarding (1)

- ESEGUO R1 = R2 + R3
- IBERNO R4 = R1 + R3
- LANCIO **R7 = R8 + R9**

Se arriviamo ad eseguire l'istruzione R4 solo per accorgerci di non poter portare a termine l'esecuzione a causa del conflitto sul registro R1, siamo costretti a bloccare la pipe. Ne scaturisce una notevole perdita di tempo, specie se l'istruzione che ha provocato il conflitto determina anche un cache miss. Per evitare questo, i processori pongono in stato di **ibernazione** istruzioni che non possono essere eseguite subito, anziché eseguirle.

Ibernazione di una istruzione: Internal Forwarding (2)

- Questa modifica nell'ordine dell'esecuzione delle istruzioni (**internal forwarding**) non dà problemi, purché le istruzioni coinvolte non agiscano sugli stessi dati.
- In questo modello quindi le istruzioni vengono prelevate in sequenza, ma non necessariamente eseguite in sequenza, badando comunque che sia sempre preservata la piena coerenza fra l'esecuzione nell'ordine naturale delle istruzioni e quella che scaturisce nella pipe in seguito ad eventuali ibernazioni.

Ibernazione di una istruzione: Internal Forwarding (3)

- R1 = MEM
- R2 = R1+R3
- R4 = R2+R7
- R8 = R9

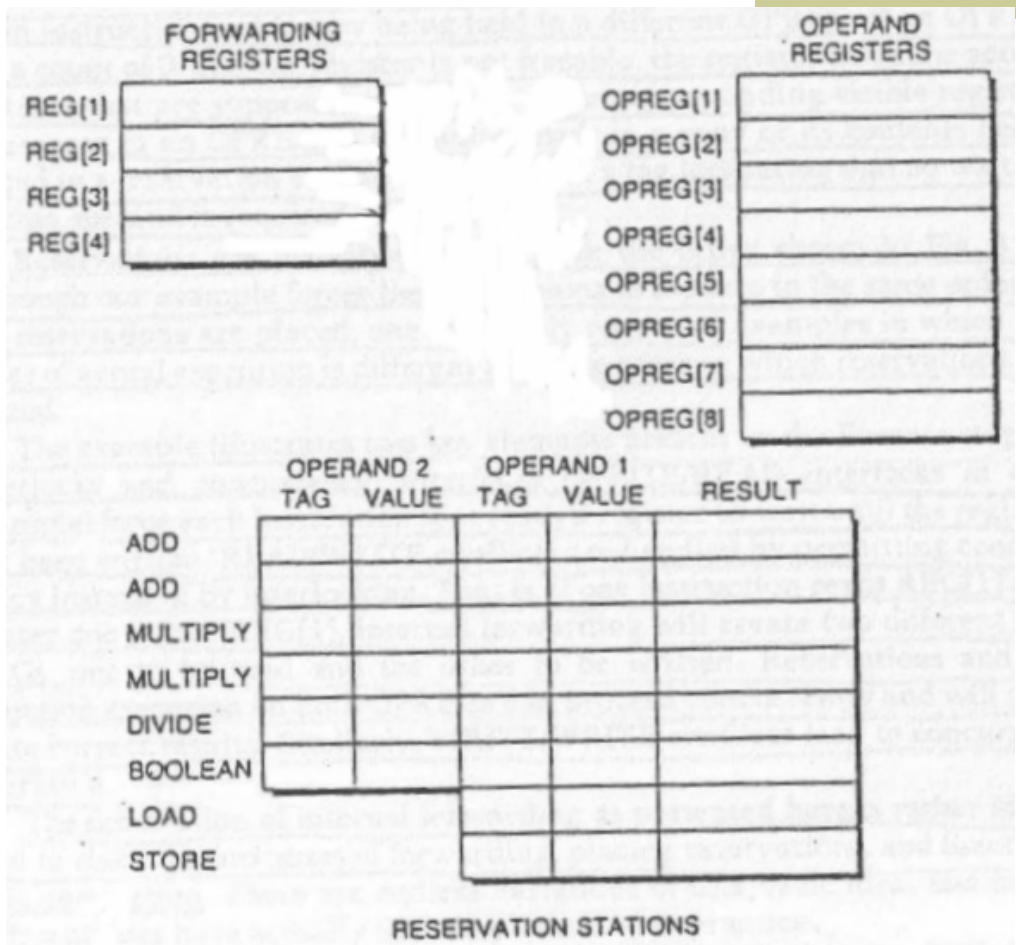
- Anche l'istruzione R4=.. dipenda dalla precedente, perché è funzione del valore di R2. In questo caso potremmo pensare di ibernare due istruzioni in fase di esecuzione anziché una.

Hw necessario per l'Internal Forwarding

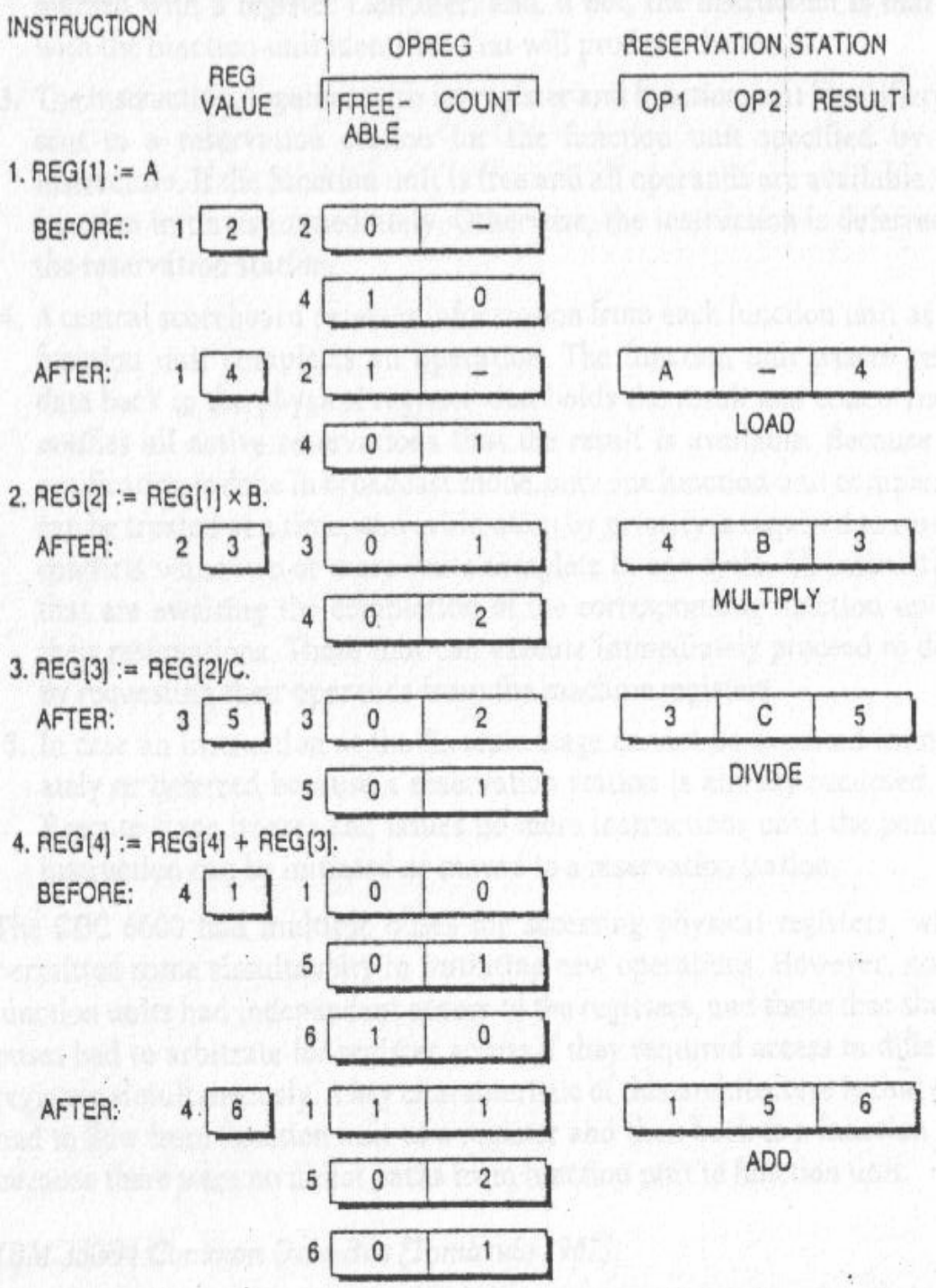
- **Forward registers** (puntatori ai registri operando),
- **Operand registers** (valori effettivi dei registri operandi),
- **Reservation Station** (tengono traccia delle istruzioni sospese).

Ibernazione di una istruzione: Internal Forwarding

```
REG[1] := A;
REG[2] := REG[1] × B;
REG[3] := REG[2]/C;
REG[4] := REG[4] + REG[3];
```



```
REG[1] := A;
REG[2] := REG[1] × B;
REG[3] := REG[2]/C;
REG[4] := REG[4] + REG[3];
```



Analisi del Forwarding: esempio

RISC:

```
LDA R1,X        R1 := X
ADD R2,R2,R1    R2 := R2 + R1
STA Y,R2        Y := R2
LDA R2,Z        R2 := Z
ADD R1,R2,R1    R1 := R2 + R1
```

Questo codice impiega:

- **7** cicli senza cache miss,
- **11** cicli Senza forwarding con 1 cache miss di 4 cicli sulla variabile X,
- **8** cicli con forwarding.

Analisi con Forwarding: 8 cicli

1. Inizia LDA R1, X – Non trova la X, cache miss. Avvia la richiesta di X verso la memoria e iberna l'istruzione.
2. Inizia ADD R2, R2, R1 – Poiché R1 non è disponibile, c'è un conflitto R/W su R1. Iberna.
3. Inizia STA Y, R2 – Poiché R2 non è disponibile, c'è un conflitto R/W su R2. Iberna.
4. Inizia LDA R2, Z – Supponiamo Z disponibile. Deve assegnare un nuovo registro a R2, quindi può eseguire questa operazione.

Analisi con Forwarding: 8 cicli

5. Inizia ADD R1, R2, R1. Poiché sono trascorsi i 4 cicli necessari, X è ora disponibile e viene caricato in R1.
6. Può eseguire la somma di cui al punto 2 (caricando R1, R2 all'ingresso dell'ALU)
7. Può eseguire la somma di cui al punto 5. Nel frattempo scrive il risultato della prima somma in Y (punto 3).
8. Scrive il risultato della seconda somma in R1.

Considerazioni

- Di fatto, la tecnica del Forwarding consiste nello effettuare una copia dei registri soltanto se i valori corrispondenti sono bloccati in lettura o in scrittura da un'altra operazione e sono coinvolti contemporaneamente in una operazione successiva.

Tecniche basate sull'analisi del codice – delayed load

```
a = b + c
d = e - f
```

```
carica b
carica c
carica e
addiziona b e c
carica f
memorizza a
sottrai f ad e
memorizza d
```

LOAD: Effetto dello stallo sulla Pipe

	Scheduling	No Scheduling
TEX	25 %	65 %
SPICE	14 %	42 %
GCC	31 %	54 %

Control Hazards (I)

Esistono diversi criteri di caricamento dei dati nella pipe, se un'istruzione da elaborare è di salto:

approccio conservativo prevede di fermare il caricamento delle istruzioni nella pipe fino alla valutazione delle condizioni di salto, eventualmente si tende ad anticipare la valutazione della condizione di salto nello stato ID.

Control Hazards (II)

approccio ottimistico non fermare la pipe ma "prevedere" l'istruzione da caricare e, in caso di scorretta previsione, ripristinare lo stato del processore (roll back).

Tecniche implementative possono essere demandate in parte al compilatore (tecniche statiche) o essere gestite direttamente dal processore (tecniche dinamiche).

Predizione dei Salti

La tecnica prevede che in parallelo alla fase di prelievo di un istruzione venga acceduta una memoria di tipo associativo (cache interna al processore) contenete l'indirizzo dell'istruzione da caricare nella fase successiva (se presente!!!).

IF	ID	EX	MEM	WB	
	IF	ID	EX	MEM	WB

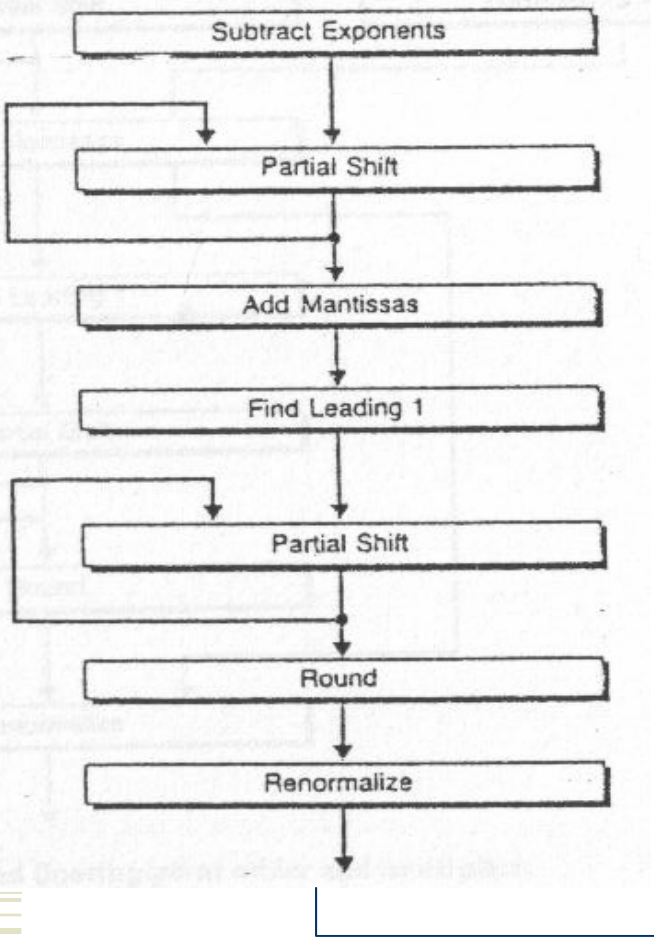
Predizione dei Salti

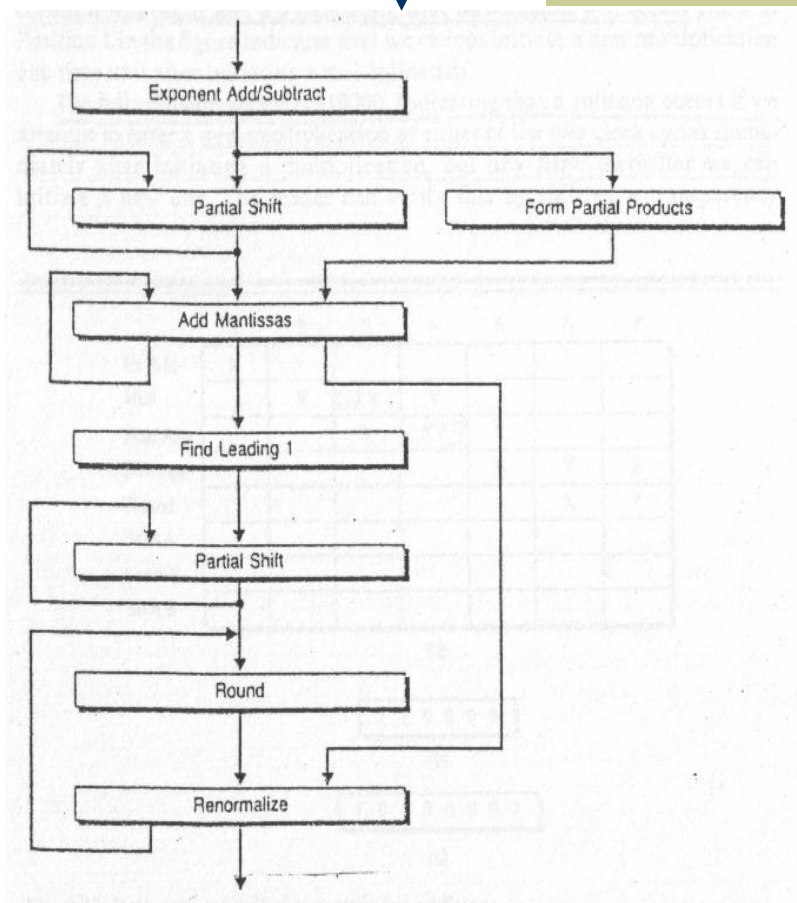
- l'indirizzo dell'istruzione da caricare è utilizzato come chiave per accedere alla memoria associativa;
- per le sole istruzioni di salto è prevista la presenza dell'indirizzo dell'istruzione da caricare nella memoria associativa;
- per le altre istruzioni l'assenza della chiave porta al caricamento della successiva istruzione calcolato a partire la PC corrente;
- sono disponibili inoltre dei bit di stato per identificare se la previsione e' ancora valida.

Architetture Superscalari

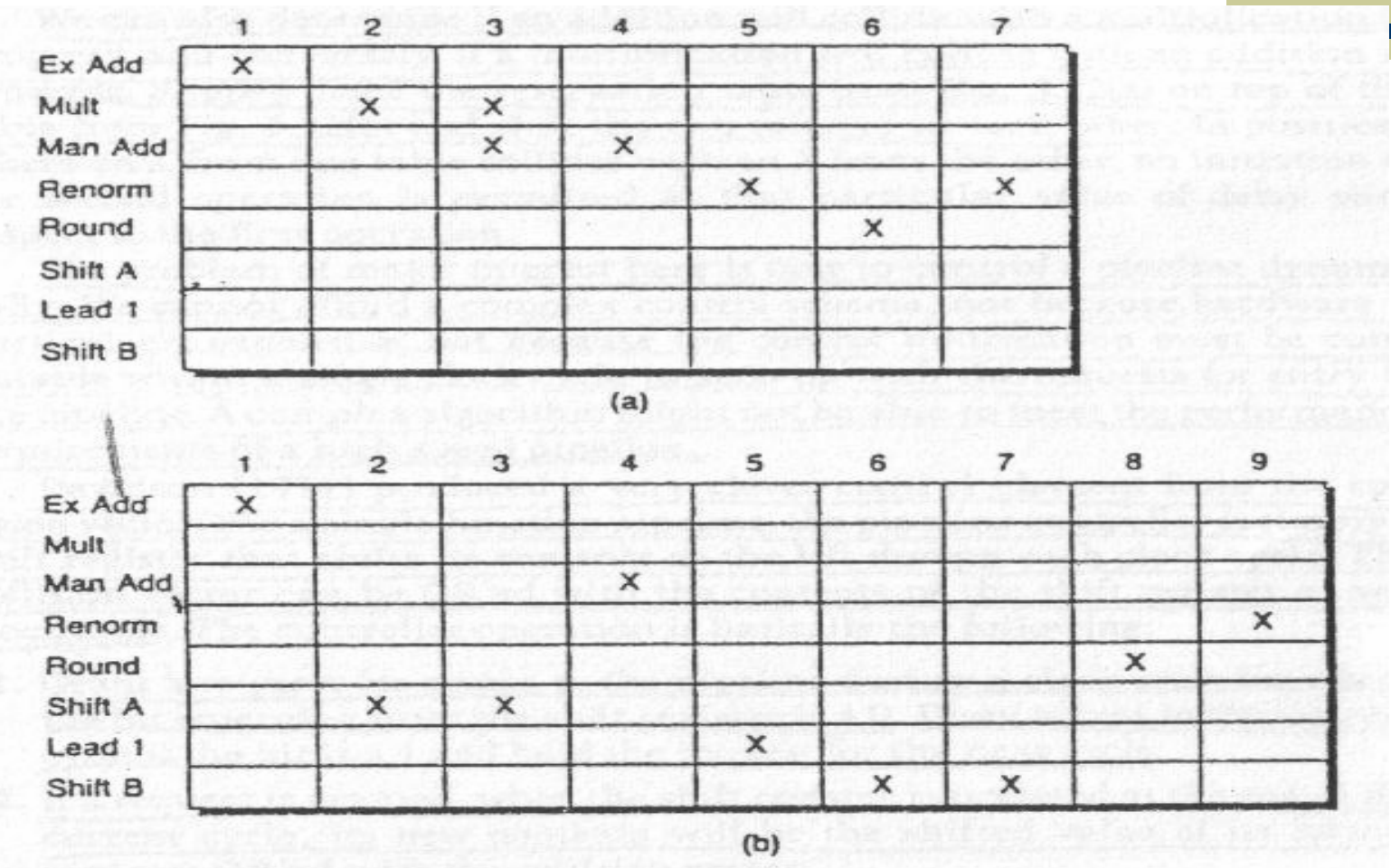
Il primo Pentium aveva due pipe, ma usava la seconda pipe solo in quei casi in cui si poteva essere sicuri che non ci fossero conflitti. Quindi non si poteva mai avere un throughput doppio. Processori moderni implementano invece vere e proprie architetture superscalari.

Collisioni nei Sistemi Superscalari con più Pipeline





Collisioni nei Sistemi Superscalari con più Pipeline



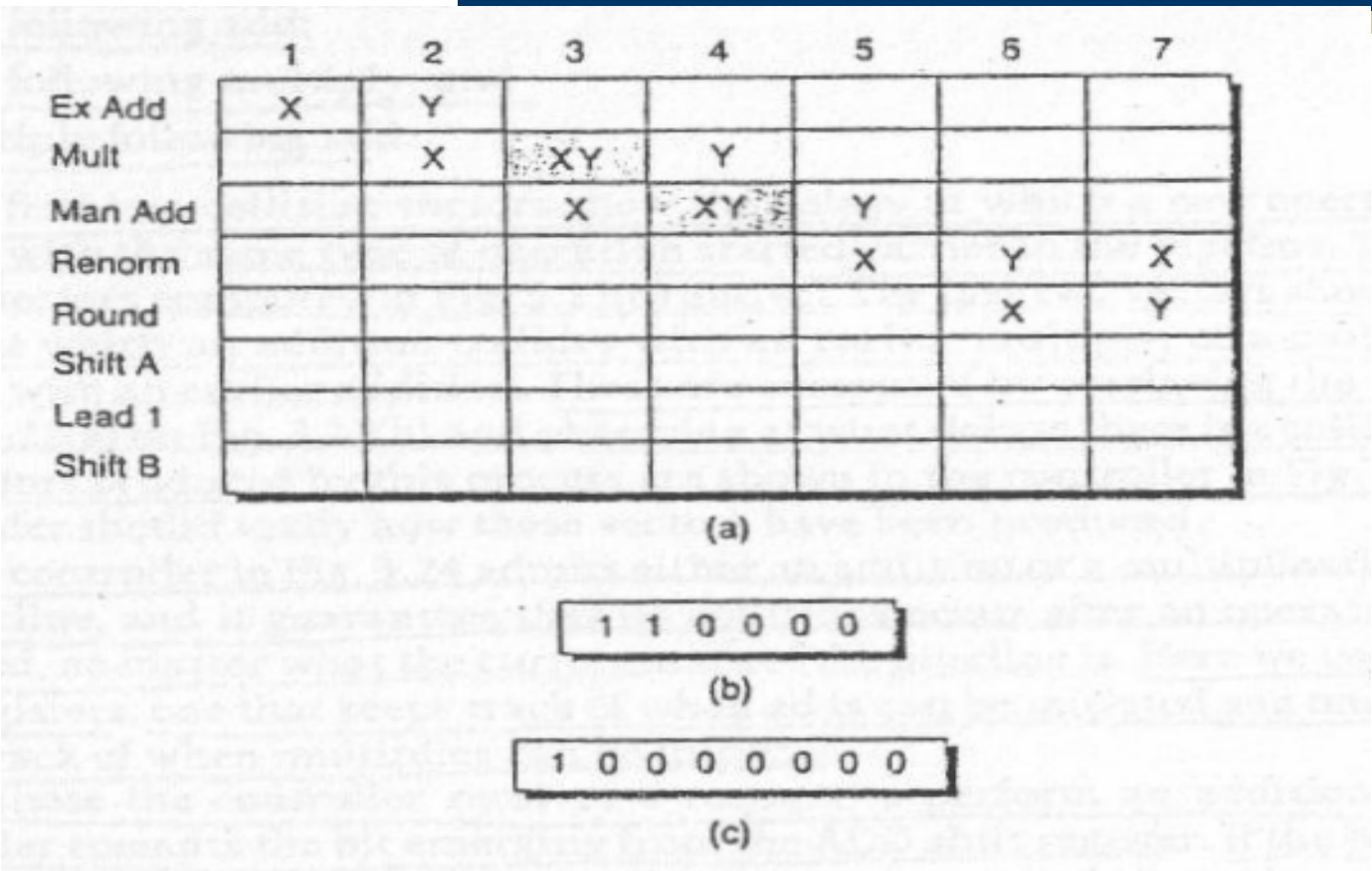
	1	2	3	4	5	6	7
Ex Add	X						
Mult		X	X				
Man Add			X	X			
Renorm					X		X
Round						X	
Shift A							
Lead 1							
Shift B							

(a)

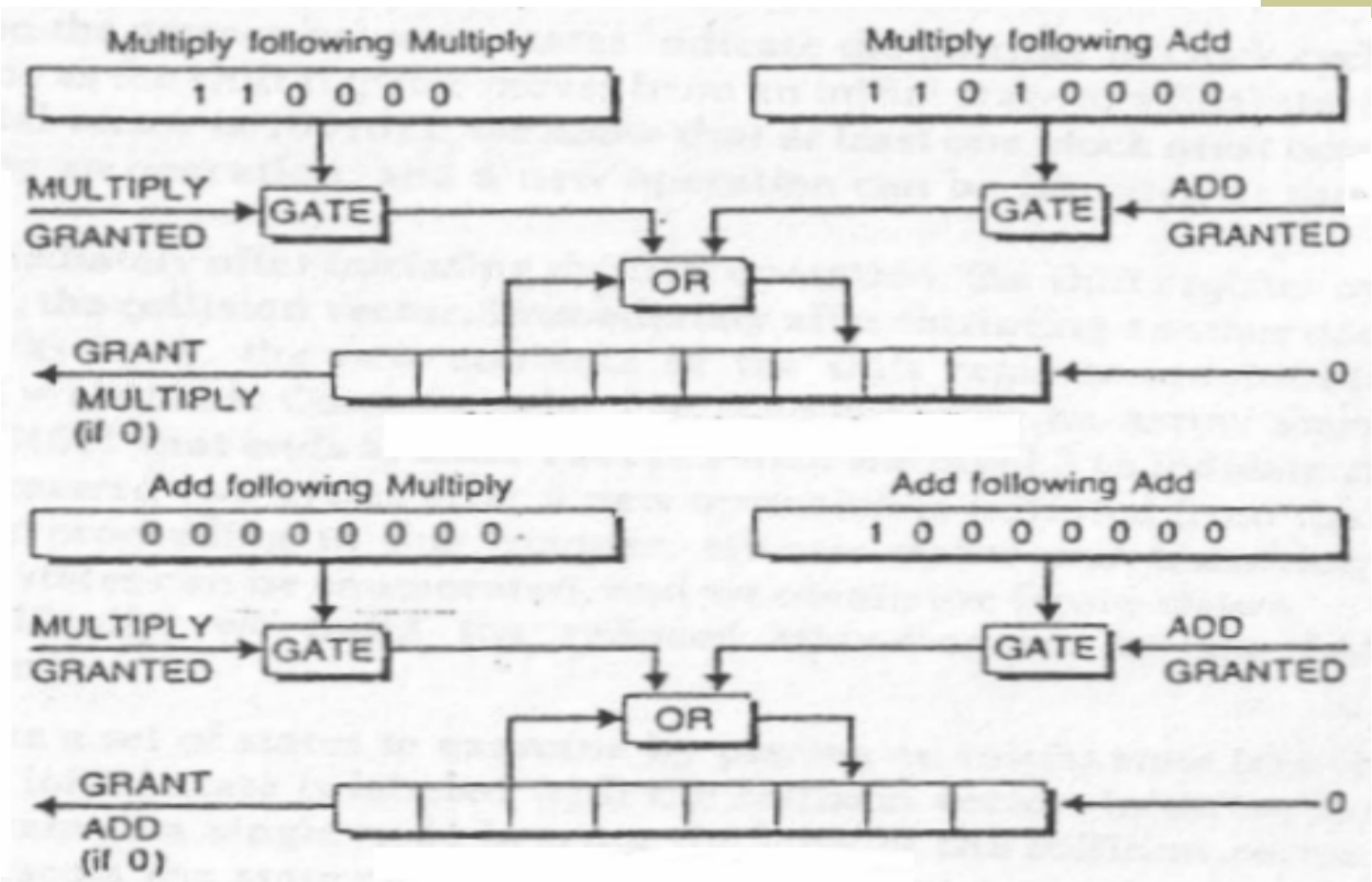
	1	2	3	4	5	6	7	8	9
Ex Add	X								
Mult									
Man Add				X					
Renorm									X
Round								X	
Shift A		X	X						
Lead 1					X				
Shift B						X	X		

(b)

Collisioni nei Sistemi Superscalari con più Pipeline



Hardware per la Gestione delle Collisioni



È meglio una macchina Cisc o una Risc?

- Un aumento della complessità del processore (Cisc), ad esempio con i vettori delle collisioni, comporta come naturale conseguenza meno spazio per la cache e quindi una cache di dimensioni ridotte.
- D'altro canto una macchina Risc esegue normalmente più istruzioni di una Cisc. Ciò comporta che la fase di Fetch venga eseguita un numero maggiore di volte, ed il ruolo della memoria diviene sempre più importante. I Risc sopperiscono a questi problemi potenziando le prestazioni delle cache.
- Nei Cisc la presenza di molti salti rallenta l'esecuzione delle istruzioni; bisogna quindi eliminare la presenza di salti non indispensabili. Questo potrebbe voler dire ad esempio che le istruzioni di un ciclo FOR con indice da 1 a 3 saranno scritte dal compilatore letteralmente 3 volte. Se si riesce a lavorare sotto questo aspetto la velocità dei sistemi Cisc diviene significativamente superiore.

- Di fatto non esistono architetture rigorosamente Cisc o Risc, ma sempre di più si assiste a sistemi ibridi che raccolgono in sé le caratteristiche di entrambe.

Advanced Pipelining

- Gestione dei salti…..
- Interruzioni……..

Riferimenti

- Patterson