


Esercizio: Dipendenze

- Che tipo di dipendenze si possono prevedere guardando questo codice sorgente?

```
if (a > c) {  
    d = d + 5;  
    a = b + d + e;  
}  
else {  
    e = e + 2;  
    f = f + 2;  
    c = c + f;  
}  
b = a + f;
```



creeranno
dipendenze dal controllo

Esercizio: Dipendenze

- Che tipo di dipendenze si possono prevedere guardando questo codice sorgente?

```
if (a > c) {  
    d = d + 5;  
    a = b + d + e;  
}  
else {  
    e = e + 2;  
    f = f + 2;  
    c = c + f;  
}  
b = a + f;
```

creeranno
dipendenze dai dati

Esercizio Pipeline : Dipendenze

Si consideri il seguente frammento di codice:

```

LOOP: LW    $1  0 ($2)      ! R1 ← mem[0+[R2]]
      ADDI  $1  $1  1      ! R1 ← [R1] + 1
      SW    $1  0 ($2)      ! mem[0+[R2]] ← [R1]
      ADD   $2  $1  $2      ! R2 ← [R1] + [R2]
      SUB   $4  $3  $2      ! R4 ← [R3] - [R2]
      BENZ  $4  LOOP        ! if([R4] != 0) PC ← indirizzo(loop)
    
```

si individuino le dipendenze **ReadAfterWrite** (RAW) e **WriteAfterWrite** (WAW).

Soluzione

#	codice	R1	R2	R3	R4	commento
1	LOOP: LW \$1, 0 (\$2)	W	R			legge R2, scrive R1
2	ADDI \$1,\$1, 1	RW				legge e scrive R1
3	SW \$1, 0(\$2)	R	R			legge R1 e R2
4	ADD \$2, \$1, \$2	R	RW			legge R1, legge e scrive R2
5	SUB \$4, \$3, \$2		R	R	W	legge R2 3 R3, scrive R4
6	BENZ \$4, LOOP				R	legge R4

Linee codice	Spiegazione dipendenza	Tipo
2←1	ADDI legge R1 che è scritto da LW	RAW
2←1	ADDI scrive R1 che è scritto da LW	WAW
3←2, 3←1	SW legge R1 che è scritto da ADDI, e prima da LW	RAW
4←2, 4←1	ADD legge R1 che è scritto da ADDI, e prima da LW	RAW
5←4	SUB legge R2 che è scritto da ADD	RAW
6←5	BENZ legge R4 che è scritto da SUB	RAW

Esercizio pipeline

Si consideri una pipeline a 4 stadi (IF, ID, EI, WO) per cui:

- i salti *incondizionati* sono risolti (identificazione salto e calcolo indirizzo target) alla fine del secondo stadio (ID)
 - i salti *condizionati* sono risolti (identificazione salto, calcolo indirizzo target e calcolo condizione) alla fine del terzo stadio (EI)
 - il primo stadio (IF) è indipendente dagli altri
 - ogni stadio impiega 1 ciclo di clock
- Si considerino le seguenti statistiche:
 - 15% delle istruzioni sono di salto *condizionale*
 - 1% delle istruzioni sono di salto *incondizionale*
 - Il 60% delle istruzioni di salto condizionale hanno la condizione soddisfatta (prese)

valutare i ritardi nella pipeline introdotti dai salti

Soluzione: valutazione dei ritardi

- Stalli per salto **incondizionato** (risolto in fase ID)

	cicli clock					
istr. eseguita	1	2	3	4	5	6
jump	IF	ID	EI	WO		
$i + 1$		IF	(qui la pipeline è "svuotata")			
istr. target			IF	ID	EI	...
$t + 1$				IF	ID	...
$t + 2$					IF	...

quindi si ha **1 ciclo** di "stallo" (non è un vero e proprio stallo: la pipeline è svuotata, quindi si esegue **1 IF inutile**)

Soluzione: valutazione delle prestazioni

- Stalli per salto **condizionato preso** (salta all'istruzione con indirizzo j)
(salto risolto in fase EI)

	cicli clock						
istr. eseguita	1	2	3	4	5	6	7
branch	IF	ID	EI	WO			
$i + 1$		IF	ID	(qui la pipeline è "svuotata")			
$i + 2$			EI	(qui la pipeline è "svuotata")			
istr. target				IF	ID	EI	WO
$t + 1$					IF	ID	EI ...

quindi si hanno **2 cicli** di "stallo" (2 fasi inutili)

Soluzione: valutazione delle prestazioni

Rappresentazione alternativa

- Stalli per salto condizionato **preso** (salta all'istruzione con indirizzo j)

		cicli clock						
		1	2	3	4	5	6	7
stadi	IF	branch	$i+1$	$i+2$	istr. target	$t+1$	$t+2$	
	ID		branch	$i+1$	bubble	istr. target	$tj+1$	
	EI			branch	bubble	bubble	istr. target	
	WO				branch	bubble	bubble	istr. target

Si noti che **ogni stadio "perde" 2 cicli di clock**:

- IF carica le istruzioni con indirizzi $i+1$ e $i+2$ che poi non terminano l'esecuzione;
- ID decodifica l'istruzione con indirizzo $i+1$ che non termina l'esecuzione e poi rimane inattiva durante il ciclo di clock 4 (**bubble**);
- EI (e successivamente WO) rimane inattiva durante i cicli di clock 4 e 5.

Soluzione: valutazione delle prestazioni

- Stalli per salto **condizionato non preso**

	cicli clock					
istr. eseguita	1	2	3	4	5	6
branch	IF	ID	EI	WO		
$i + 1$		IF	ID	EI	...	
$i + 2$			IF	ID	...	
$i + 3$				IF	...	

quindi si hanno **0 cicli** di “stallo”

Soluzione: valutazione delle prestazioni

valutare i ritardi nella pipeline introdotti dai salti:

- per ogni salto incondizionato: 1 ciclo di ritardo
- per ogni salto condizionale preso: 2 cicli di ritardo
- per ogni salto condizionale non preso: 0 cicli di ritardo

Probabilità di eseguire una delle istruzioni di salto

salto incondizionato → **0,01** perché 1 su 100 è un salto incondizionato
 salto condizionale preso → $0,15 \cdot 0,6 = \mathbf{0,09}$ perché 15 istr. su 100, e il 60% salta
 salto condizionale non preso → $0,15 \cdot 0,4 = \mathbf{0,06}$ perché 15 istr. su 100 e il 40% non salta

la frazione di cicli in cui si ha stallo/ritardo è:

$$\begin{aligned}
 &\text{prob_jump} * \text{stalli_jump} && \mathbf{[0,01*1]} \\
 &+ && + \\
 &\text{prob_branch_preso} * \text{stalli_branch_preso} && \mathbf{[0,09*2]} \\
 &+ && + \\
 &\text{prob_branch_non_preso} * \text{stalli_branch_non_preso} && \mathbf{[0,06*0] = 0,19}
 \end{aligned}$$

Soluzione: valutazione delle prestazioni

fattore di velocizzazione di una pipeline a k stadi, a regime, in funzione del numero di stalli:

$$S_k = \frac{1}{1 + \text{frazione_cicli_stallo}} k$$

$$S_k = \frac{1}{1 + 0,19} 4 = 3,36$$

quanto più veloce, a regime, sarebbe la pipeline senza gli stalli introdotti dai salti?

$$\frac{\text{prestazione di pipeline **senza** stalli}}{\text{prestazione di pipeline **con** stalli}} = 4 / 3,36 = 1,19$$

Architetture RISC





Reduced Instrucion Set Computer

Caratteristiche chiave

Le architetture RISC sono caratterizzate da

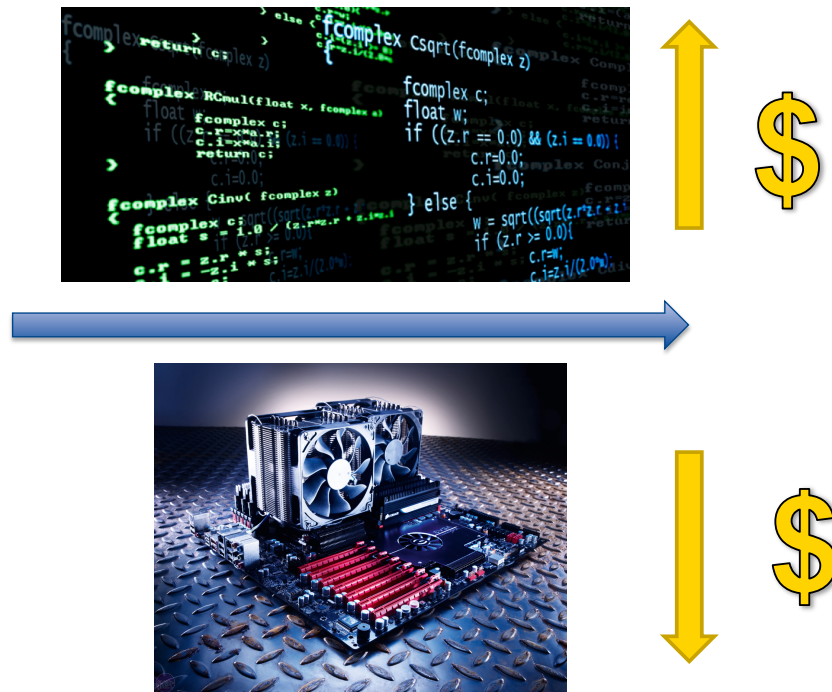
1. un grande numero di **registri general-purpose**, e/o l'uso di un compilatore che ottimizza l'uso di questi registri
2. un **set istruzioni semplice e limitato**
3. ottimizzazione della **pipeline** (basata sul formato fisso delle istruzioni, modi di indirizzamento semplici,...)

Comparazione fra vari processori

	Complex Instruction Set (CISC) Computer			Reduced Instruction Set (RISC) Computer	
Characteristic	IBM 370/168	VAX 11/780	Intel 80486	SPARC	MIPS R4000
Year developed	1973	1978	1989	1987	1991
 Number of instructions	208	303	235	69	94
 Instruction size (bytes)	2-6	2-57	1-11	4	4
 Addressing modes	4	22	11	1	1
 Number of general-purpose registers	16	16	8	40 - 520	32
Control memory size (Kbits)	420	480	246	—	—
Cache size (KBytes)	64	64	8	32	128

Evoluzione

- Co-evoluzione tra hardware e linguaggi di programmazione



Evoluzione

- **High-level languages**
 - permettono di esprimere l'algoritmo risolutivo in modo più conciso: **cosa**
 - lasciano al compilatore il compito di gestire i dettagli : **come**
 - supportano costrutti di programmazione strutturata: paradigmi *imperativo, funzionale, logico, object-oriented*

