

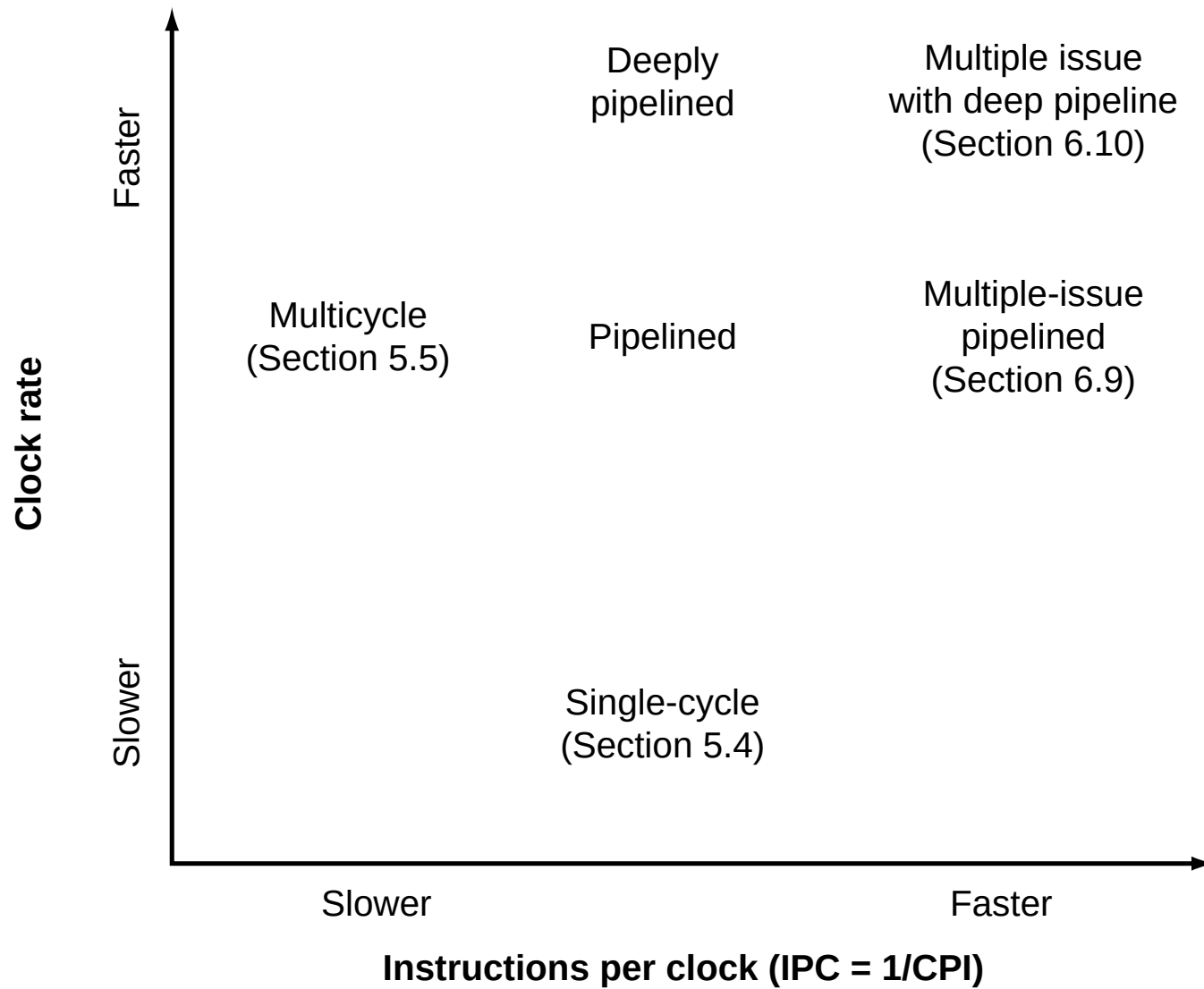
Calcolatori Elettronici B

a.a. 2006/2007

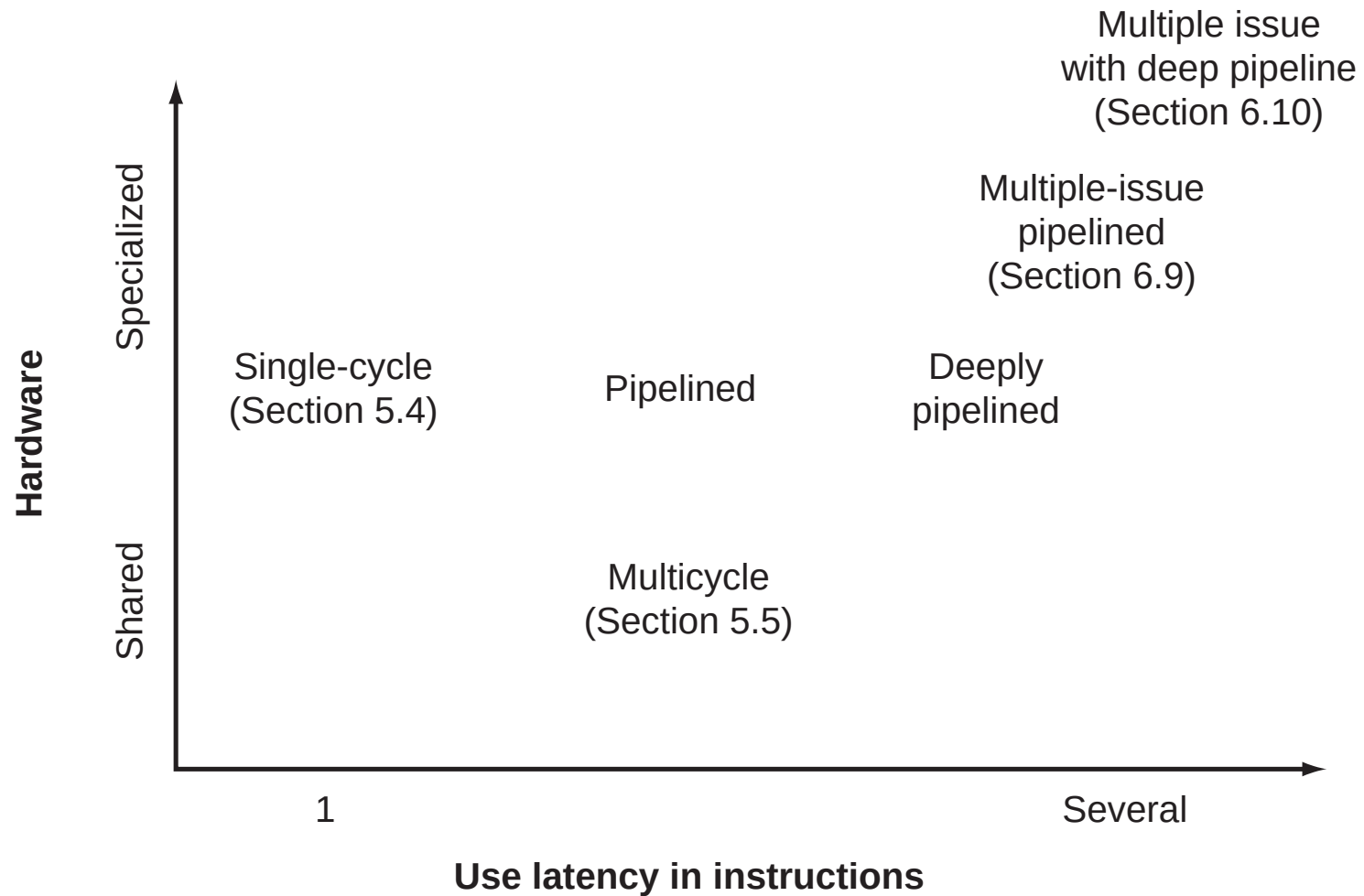
**PIPELINE:
CONCLUSIONI**

Massimiliano Giacomini

Periodo di clock e CPI



Specializzazione hardware e “use latency”



Pipeline: influenza su struttura delle istruzioni

La struttura delle istruzioni è influenzata dalla struttura dell'hardware per quanto riguarda:

- effetti collaterali
 - modalità di indirizzamento
 - gestione dei codici di condizione

Effetti Collaterali

- Quando l'esecuzione dell'istruzione cambia il contenuto di registri diversi dal registro destinazione.

Esempi di effetti collaterali

1. Istruzioni con autoincremento (autodecremento) -- NB: reg. destinazione alla fine
M68000 **MOVE (A0)+, D0**
2. Uso stack (in cui il puntatore viene incrementato/decrementato)
3. Uso dei flag di condizione
M68000 **MOVE #50, D3**
 ADD D1, D2
 ADDX D3, D4

Dipendenza: esplicita tra **MOVE** e **ADDX**

implicita tra **ADD** e **ADDX**

X è flag che indica il riporto.

Istruzioni con effetti collaterali: generano dipendenze multiple e “indirette”

⇒ aumento complessità dell'HW per propagazione e stallo

⇒ aumento delle penalità dovute alle dipendenze implicite

⇒ aumento della complessità del SW:

più difficile per il compilatore riordinare le istruzioni
con molte dipendenze



In pipeline: è bene ridurre gli effetti collaterali

[solo il registro destinazione dovrebbe essere modificato
dall'esecuzione di una istruzione]

Tuttavia, istruzioni con effetti collaterali sono utili in molti casi

(p.es. indirizzamento con autoincremento è utile per la gestione dei vettori,
oppure flag di condizione utili per prendere decisioni ecc.)

Indirizzamento e pipeline

- Modalità di indirizzamento complesse (con autoincremento-autodecremento, con indice, indiretta + combinate) sono utili...
- Occorre però tenere presente che maggior complessità può portare a HW più complicato (aumentando T_{clock}) e performance peggiori...
- Nel caso della pipeline le complicazioni comportate sono ancora maggiori

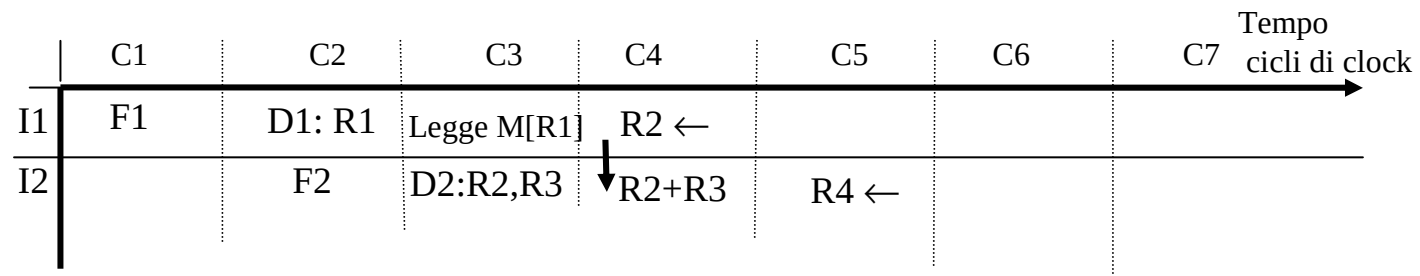


Scelte di compromesso: occorre anche tener conto di quanto i compilatori usano le varie modalità di indirizzamento

Indirizzamento indiretto a memoria

I1: Load R2, (R1)

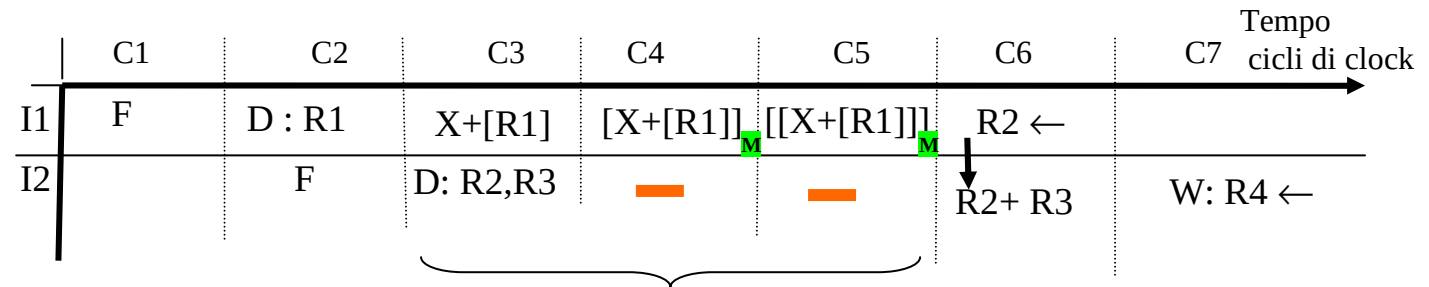
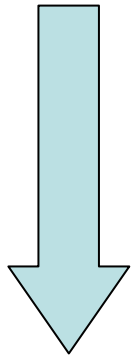
I2: Add R4, R2,R3



No stallo!

I1: Load R2, (X (R1))
 I2: Add R4, R2,R3

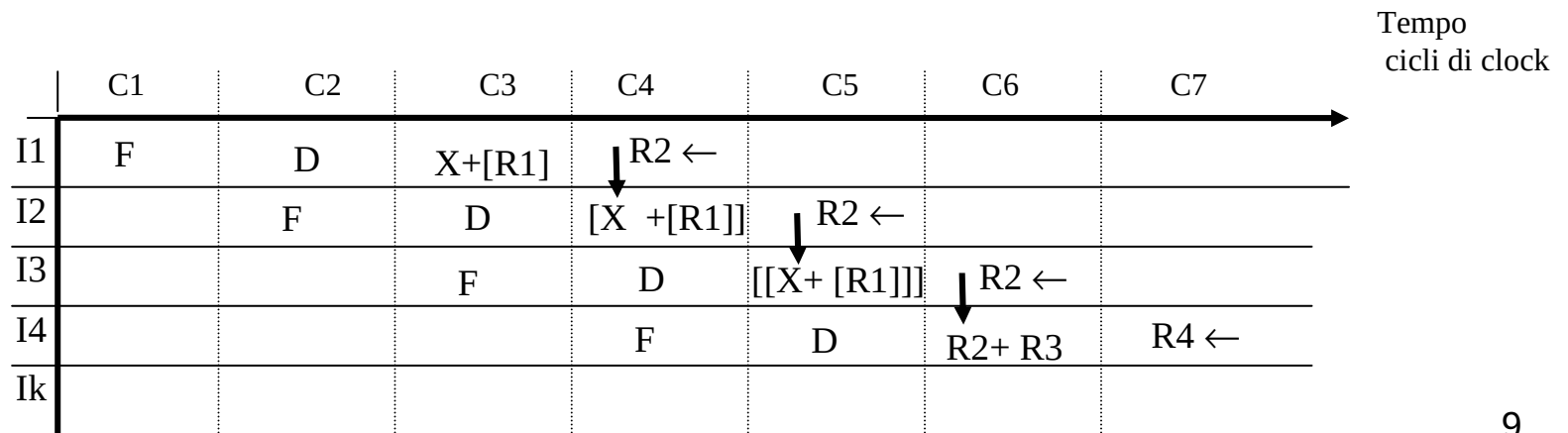
Indirizzamento complesso...



I1 richiede più cicli di clock $\Rightarrow$ stallo

I1: Add R2, #X,R1
 I2: Load R2, (R2)
 I3: Load R2, (R2)
 I4: Add R4, R2,R3

... vs. Indirizzamento semplice



Quindi una modalità di indirizzamento semplice:

- Spende lo stesso numero di cicli
[poiché non impegna per più cicli una unità funzionale,
ciascuna istruzione tende a provocare meno stalli]
- Hardware più semplice (può permettere di ridurre ciclo di clock)
- Modalità di indirizzamento semplice trattabili più
facilmente dai compilatori
- [Richiede però più memoria...]

QUINDI...

Modalità di Indirizzamento più adatte alla pipeline

- l'accesso ad un operando non richiede più di un accesso a memoria
- solo load e store accedono ad operandi in memoria
- le tecniche di indirizzamento non hanno effetti collaterali

Tipici modi usati nel pipeline:

a registri, indiretto a registri, indicizzato (somma offset e valore registro) e relativo (indicizzato al PC)

Questi indirizzamenti sono alla base del RISC
(ma oggi si tende ad essere meno rigidi)

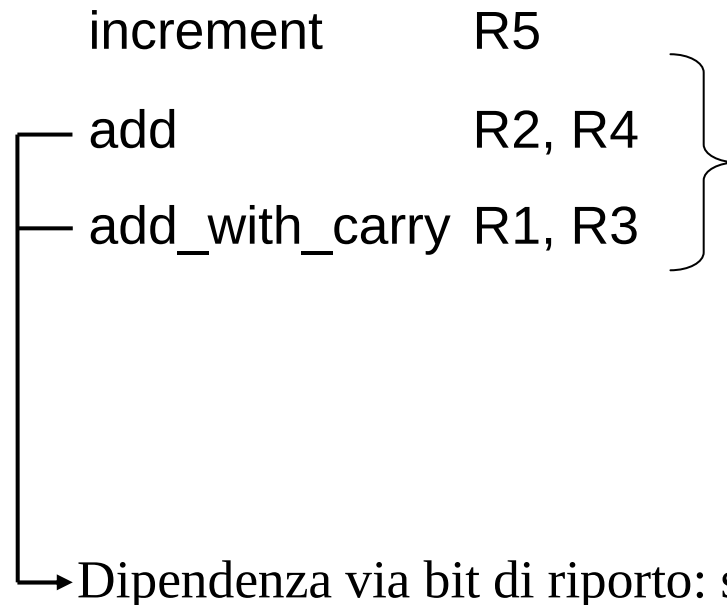
↑
**o contenuto altro
registro o immediato**

e.g. PowerPC: alcune forme di indicizzato permettono l'aggiornamento dei registri indice (effetto collaterale!)
Non è né RISC né CISC

Codici di condizione

- Le dipendenze tra istruzioni causate dai codici di condizione (istruzione il cui comportamento dipende da un flag modificato da un'istruzione precedente) devono essere gestite dal compilatore e dall'HW:
 - il **compilatore** deve riordinare istruzioni per evitare stalli, ma la sua flessibilità è ridotta dalle dipendenze causate tra flag
[non è possibile il riordino se il risultato cambia]
 - l'**HW** deve riconoscere le dipendenze per forzare eventualmente lo stallo

Caso tipico



**Queste due eseguono
operazioni in doppia
precisione: il riporto della
prima è usato nella
seconda**

**R1-R2+
R3-R4**

Compilatore: dovrebbe tentare di inserire istruzioni tra le due;
non può utilizzare increment perché altera bit di riporto!

Hardware: deve riconoscere che add_with_carry dipende da add
e ritardare l'esecuzione della seconda se necessario
(ovviamente, è sempre valida la possibilità di propagazione
per risparmiare qualche ciclo di stallo)



Regole da seguire per l'architettura nel caso di pipeline...

Gestione dei codici di condizione

1. Rendere l'uso dei codici di condizione facili da individuare per i circuiti di decodifica (unità di controllo)
e.g. PowerPC utilizza due bit nell'istruzione per individuare i codici di condizione da essa modificati
2. I flag dovrebbero essere affetti dal minimo numero di istruzioni possibili

Nell'Assembler PowerPC ci sono istruzioni separate per la gestione dei flag

ADDI	R5, R5, 1	- non modifica flag
ADDC	R4, R2, R4	- modifica il flag (in un registro di stato)
ADDE	R3, R1, R6	- usa il carry flag come sorgente (memorizzato in un registro di stato)

Se il compilatore deve riordinare le istruzioni, può inserire ADDI tra le due istruzioni ADDC e ADDE (che dipendono dal flag): se la ADDE avesse bisogno di uno stallo per aspettare la produzione del riporto da ADDC, posso ridurre o eliminare la perdita di cicli tramite riordinamento.

CONCLUSIONI

Il potenziale della pipeline si sfrutta con un accurato progetto di:

- Set delle istruzioni
 - Modi di indirizzamento limitati
 - Esplicitare l'uso dei codici di condizione
 - Regolarità del formato delle istruzioni
- Hardware (controllo e data path)
 - In particolare, compromesso nel numero di stadi:
[aumento $\Rightarrow$ $\uparrow$ throughput ideale, ma anche $\uparrow$ probabilità di stallo dovuto a dipendenze di istruzioni distanti e inoltre aumento $\Rightarrow$ $\uparrow$ costi]
- I compilatori
 - possono tener conto dell'HW per produrre codice ottimizzato