


La pipeline

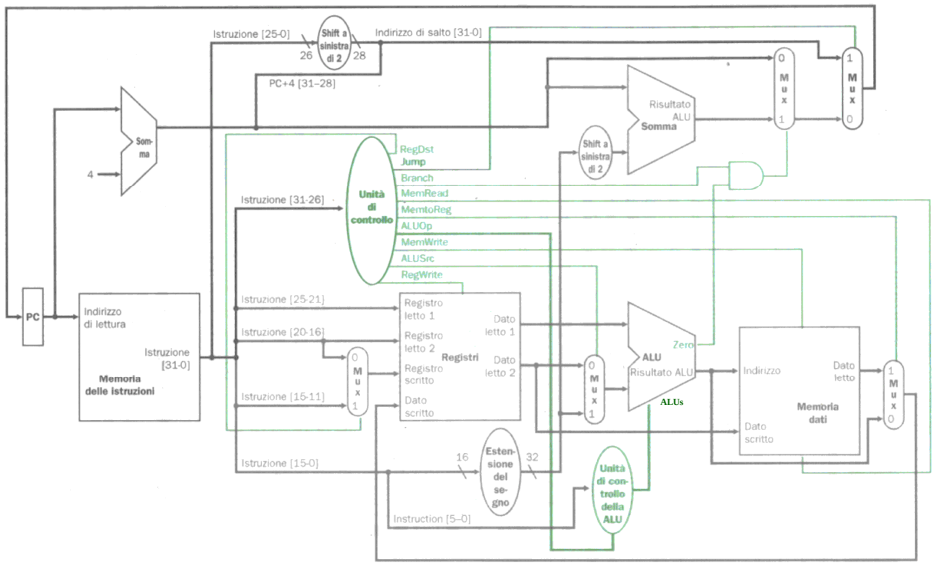
Prof. Alberto Borghese
Dipartimento di Scienze dell'Informazione
alberto.borghese@unimi.it

Università degli Studi di Milano

Riferimento al Patterson edizione 5: 4.5 e 4.6


Sommario

The control and data path



Osservazioni

Il ciclo di esecuzione di un'istruzione si compie in un unico ciclo di clock.

⇩

Ogni unità funzionale può essere utilizzata 1 sola volta.

⇩

Duplicazione Memoria: Memoria dati e memoria istruzioni.
Triplicazione ALU: 3 ALU: 2 sommatori + 1 general purpose.

⇩

Il periodo del clock è determinato dal cammino critico dell'istruzione più lunga: lw.
Utilizzare un clock per ogni fase di esecuzione porterebbe ad un tempo di esecuzione variabile per le diverse istruzioni: beq = 3 cicli di clock, aritmetche e sw, 4 cicli di clock, lw 5 cicli di clock. Si può fare di meglio => pipeline.

Intuizione della pipeline

Anna, Bruno, Carla e Dario devono fare il bucato.

Devono lavare, asciugare, piegare e mettere via ciascuno un carico di biancheria (4 stadi per la lavorazione del bucato)

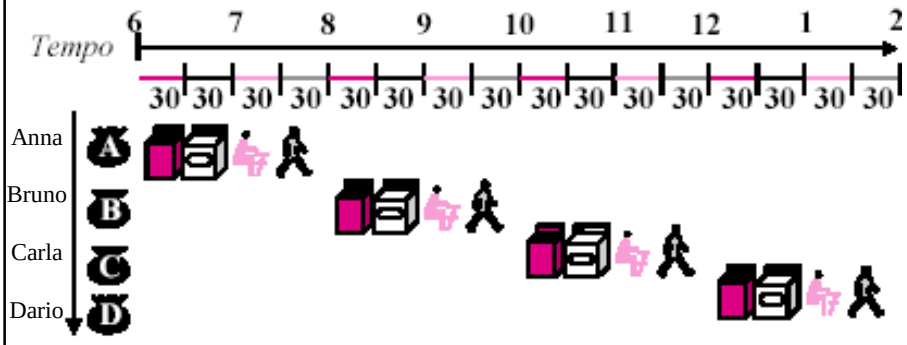
La lavatrice richiede 30 minuti.

L'asciugatrice richiede 30 minuti.

Stirare richiede 30 minuti.

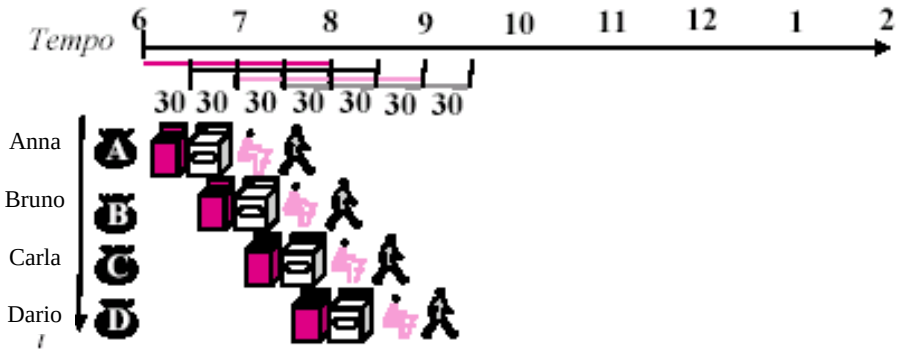
Piegare e mettere via richiede 30 minuti.

La lavanderia sequenziale



In totale vengono richieste 8 ore.

Lavanderia con pipeline



In totale vengono richieste 3.5 ore.

Osservazioni sulla pipeline

Il tempo di ciascuna operazione elementare non viene ridotto.

Gli stadi della pipe-line lavorano in contemporanea perché utilizzano unità funzionali differente.

Le unità funzionali lavorano sequenzialmente (in passi successivi) su istruzioni successive.

Tutto il materiale che serve per il passo successivo viene prelevato dal passo precedente.

Viene aumentato il throughput.

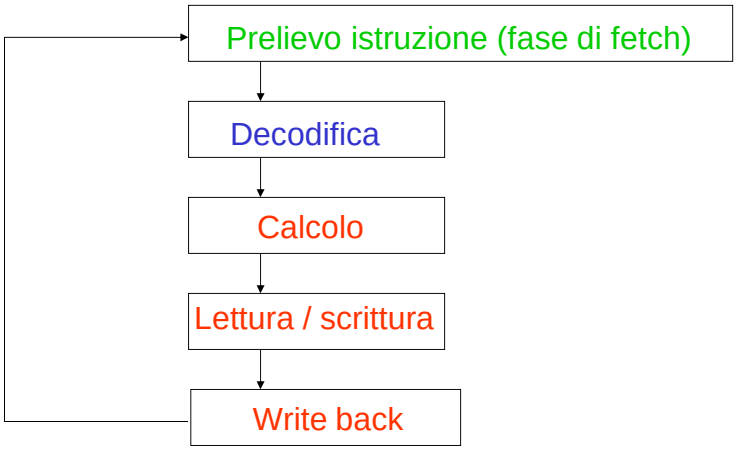
Sommario

Introduzione sulla pipeline

Gli stadi della pipeline

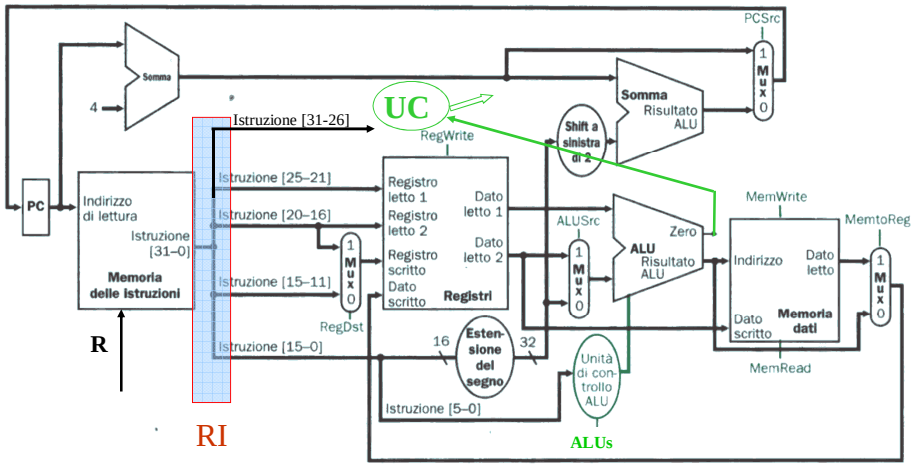
Rappresentazione del flusso di esecuzione in una pipeline

Ciclo di esecuzione di un'istruzione MIPS



Le istruzioni richiedono 5 passi → 5 stadi della pipe-line

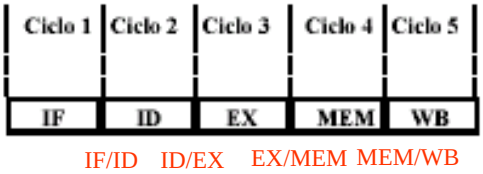
Schema generale CPU a ciclo singolo



Utilizzo unità funzionali della CPU

Passo esecuzione	ALU (+4)	Mem Instr.	Register File R	ALU (Pc+Offset)	ALU	Mem Dati	Register File W
Fase fetch	Yes	Yes	N0	N0	N0	N0	N0
Decodifica	N0	N0	Yes	N0	N0	N0	N0
Exe – beq	N0	N0	N0	Yes	Yes (diff)	N0	N0
Exe – j	N0	N0	N0	N0	N0	N0	N0
Exe – R	N0	N0	N0	N0	Yes	N0	N0
Exe sw / lw	N0	N0	N0	N0	Yes	N0	N0
Mem – lw/sw	N0	N0	N0	N0	N0	Yes R/W	N0
Mem – altre	N0	N0	N0	N0	N0	N0	N0
WB – R	N0	N0	N0	N0	N0	N0	Yes
WB – lw	N0	N0	N0	N0	N0	N0	Yes
WB sw/beq	N0	N0	N0	N0	N0	N0	N0

I 5 stadi della pipeline

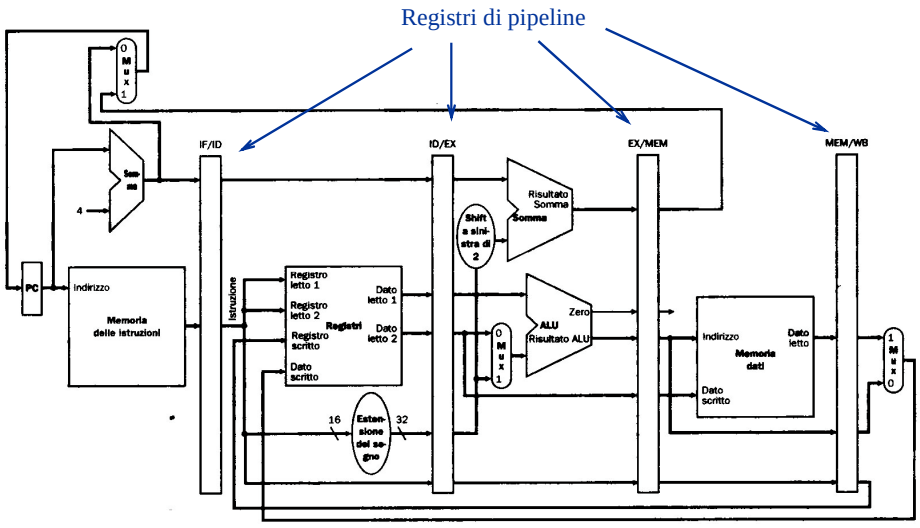


Tra due cicli sono posti dei registri denominati registri di pipe-line.

Nomi degli stadi di pipeline:
- IF: Prelievo istruzione (Instruction fetch)
- ID: Decodifica istruzione (+lettura register file)
- EX: Esecuzione
- MEM: Accesso a memoria (lettura/scrittura)
- WB: Scrittura del register file

NB Uno stadio inizia il suo lavoro quando il clock va basso e trasferisce in quello stadio l'elaborazione effettuata dallo stadio precedente

CPU con pipeline



Il ruolo dei registri

Ciascuno stadio produce un risultato. La parte di risultato che serve agli stadi successivi deve essere memorizzata in un registro.

Il registro mantiene l'informazione anche se lo stadio precedente riutilizza l'unità funzionale. Esempio: l'istruzione letta viene salvata nel registro IF/ID (cf. Instruction Register), nella fase IF posso leggere un'altra istruzione.

Il registro mantiene tutte le informazioni che servono per l'esecuzione dello stadio presente fino all'esecuzione della fase di WEB. Ad esempio il numero del registro di scrittura viene selezionato nella fase EX e viene portato avanti fino alla fase di WB.

La UC della CPU con pipeline

Definizione dei segnali di controllo per ogni stadio, per ogni istruzione.

Definizione dell'UC in grado di generare correttamente questi segnali.

Segnali di controllo su 1 bit

Nome segnale	Effetto quando è negato	Effetto quando è affermato
RegDst	Il numero del registro destinazione proviene dal campo rt (R2, bit 20-16)	Il numero del registro destinazione proviene dal campo rd (bit 15-11)
RegWrite	Nessuno	Nel registro specificato all'ingresso registro scritto del Register File, viene scritto il valore presente all'ingresso Dato Scritto
ALUSrc	Il secondo operando della ALU proviene dalla seconda uscita in lettura del Register File	Il secondo operando della ALU è la versione estesa (con segno) del campo offset
Branch	Il valore del PC viene sostituito dall'uscita del sommatore che calcola PC+4 (condizionato all'uscita di ALU)	Il valore del PC viene sostituito dall'uscita del sommatore che calcola la destinazione del salto (condizionato all'uscita di ALU)
MemRead	Nessuno	Il contenuto della cella di memoria dati indirizzata dal MAR è posto nel MDR
MemWrite	Nessuno	Il contenuto in ingresso al MDR, viene memorizzato nella cella il cui indirizzo è caricato nel MAR
MemtoReg	Il valore inviato all'ingresso Dato al Register File proviene dalla ALU	Il valore inviato all'ingresso Dato al Register File proviene dalla memoria

Scrittura PC e scrittura dei registri di pipeline ad ogni fronte di clock (ad ogni stadio).

Osservazioni

Il contenuto di *rt* ed il numero di scrittura nel Register File (*rd*) vengono portati attraverso i vari stadi.

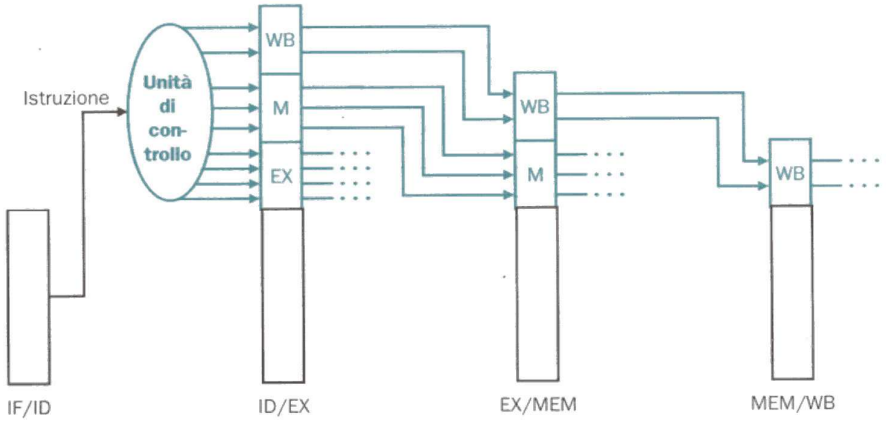
Nella fese di fetch e di decodifica non esistono segnali di controllo particolari.

I segnali di controllo particolari (legati alle diverse istruzioni) si possono così raggruppare:

	Exec				Memory			WB	
Istruzione	Reg Dst	ALUs1	ALUs0	ALU Src	Branch	Mem Read	Mem Write	Reg Write	Mem2 Reg
Format-R	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X

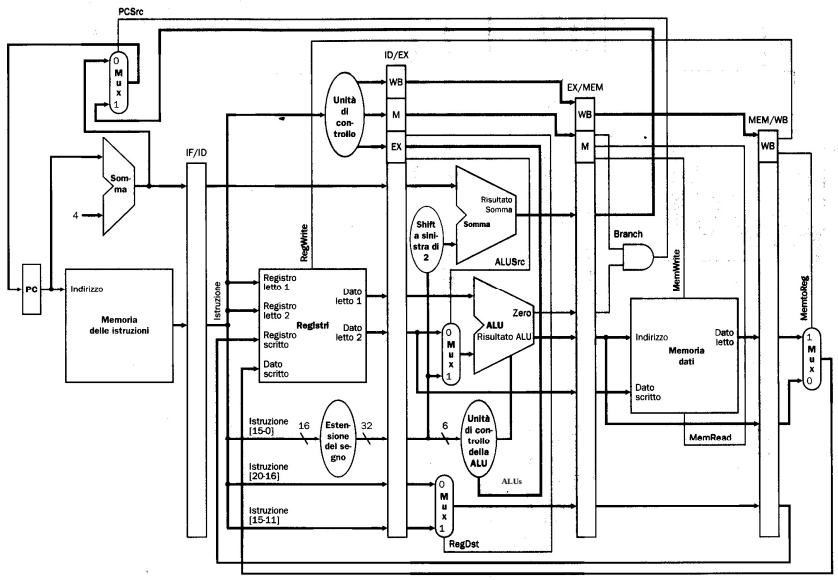
Generazione dei segnali di controllo

I segnali di controllo vengono generati nello stadio di decodifica e propagati.



Viene separata la fase di generazione dei segnali di controllo dalla fase di utilizzo.

UC per pipeline



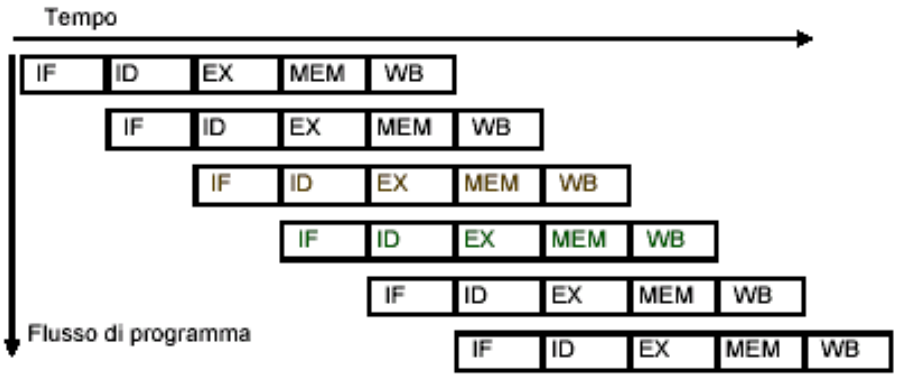
Sommario

Introduzione sulla pipeline

Gli stadi della pipeline

Rappresentazione del flusso di esecuzione in una pipeline

Rappresentazione convenzionale della pipeline



MIPS e pipeline

Fase di fetch semplificata: tutte le istruzioni hanno la stessa lunghezza.

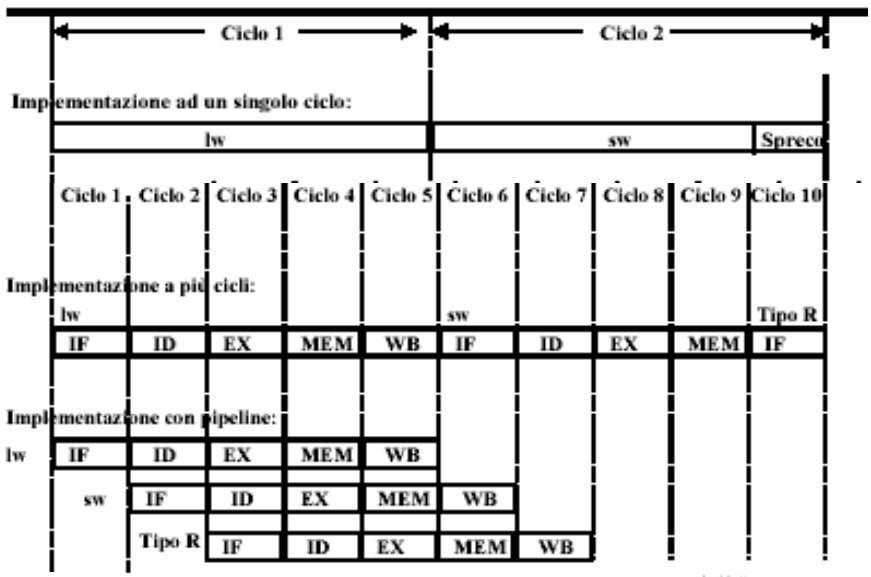
Numero ridotto di formati, i registri sono sempre nella stessa posizione (si può decodificare il codice operativo e leggere i registri).

Non ci sono operazioni sui dati in memoria: se utilizzo la ALU per effettuare dei calcoli, non dovrò accedere alla memoria. Se utilizzo la ALU per calcolare l'indirizzo, accederò alla memoria nell'istante successivo.

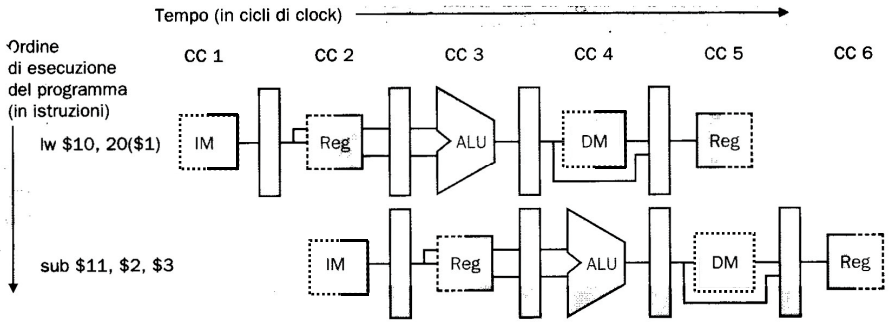
Allineamento delle istruzioni al byte.

Su architetture CISC la pipeline sarebbe più complicata...., ma vedremo che le gerarchie di memoria aiutano a semplificare il problema.

I vantaggi della pipeline



Visualizzazione grafica di una pipeline



Rappresentazione grafica di una istruzione di add in pipeline

Esempio: add $s0, $t0, $t1

I rettangoli grigi a destra indicano lettura, a sinistra indicano scrittura. I componenti bianchi, indicano il loro non utilizzo.

Esempio: lw $t0, 20($t1)

Esempio: sw $s0, 20($s1)

Miglioramento delle prestazioni

Il miglioramento massimo è una riduzione del tempo di un fattore pari al numero di stadi della pipe-line.

Nell'esempio precedente (2 istruzioni di lw), il tempo richiesto con la pipe-line è di 12ns contro i 20ns senza pipe-line. Il miglioramento teorico, prevedrebbe 4ns. Allora?

Throughput!! Miglioramento relativo al lavoro globale (con pipe-line senza stalli, 2ns ad istruzione)

Sommario

Introduzione sulla pipeline

Gli stadi della pipeline

Rappresentazione del flusso di esecuzione in una pipeline