


La pipeline

Prof. Alberto Borghese
Dipartimento di Informatica
alberto.borghese@unimi.it

Università degli Studi di Milano

Riferimento al Patterson edizione 5: 4.5 e 4.6

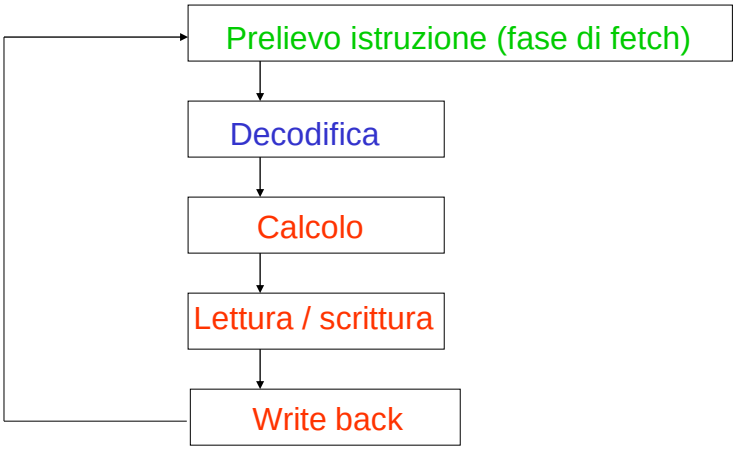

Sommario

I problemi di un'implementazione a singolo ciclo

La pipeline

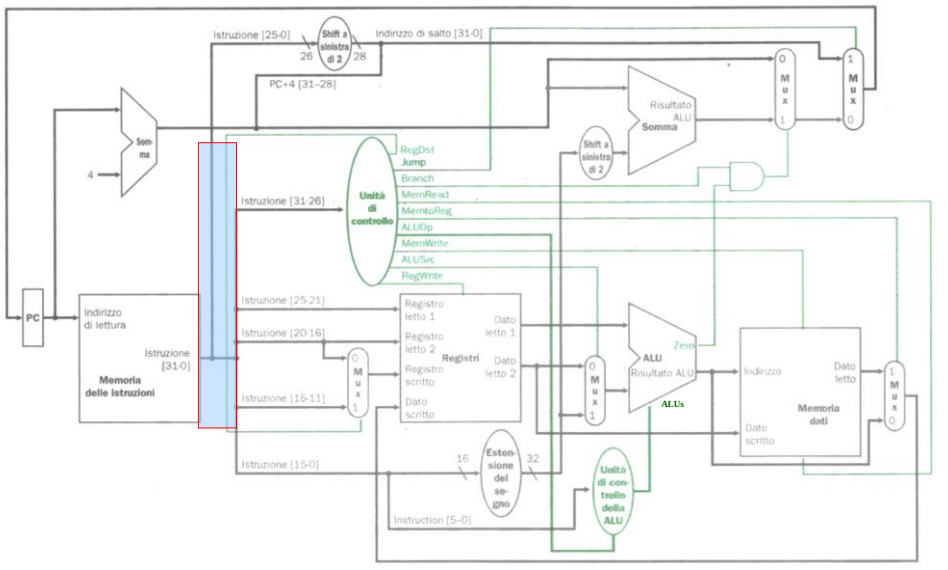
L'unità di controllo della pipeline

Ciclo di esecuzione di un'istruzione MIPS

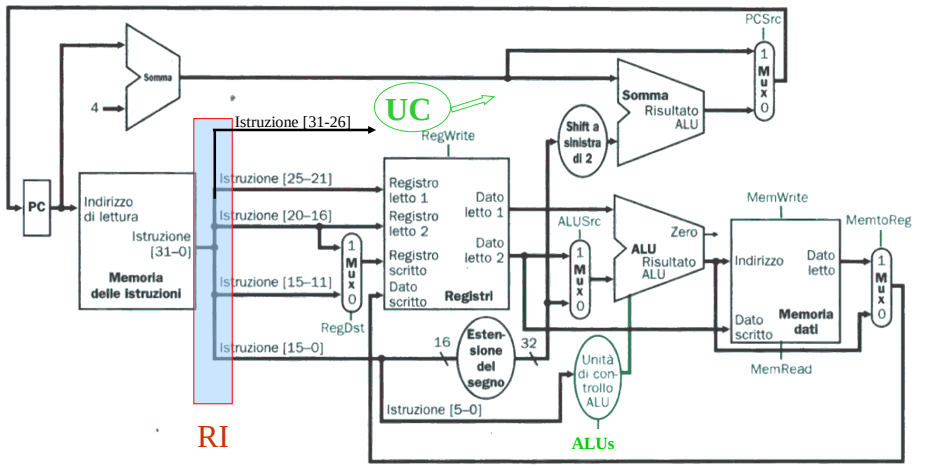


Le istruzioni richiedono 5 passi → Ciclo di clock dimensionato su 5 passi

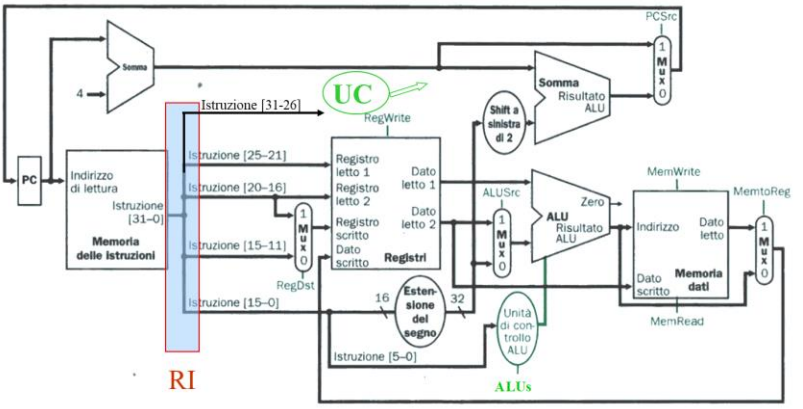
CPU a singolo ciclo di clock



Schema generale CPU a ciclo singolo
datapth

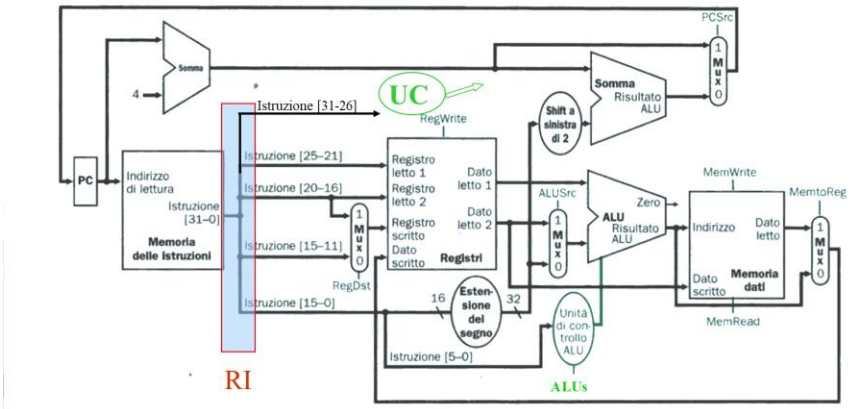


Fase di fetch



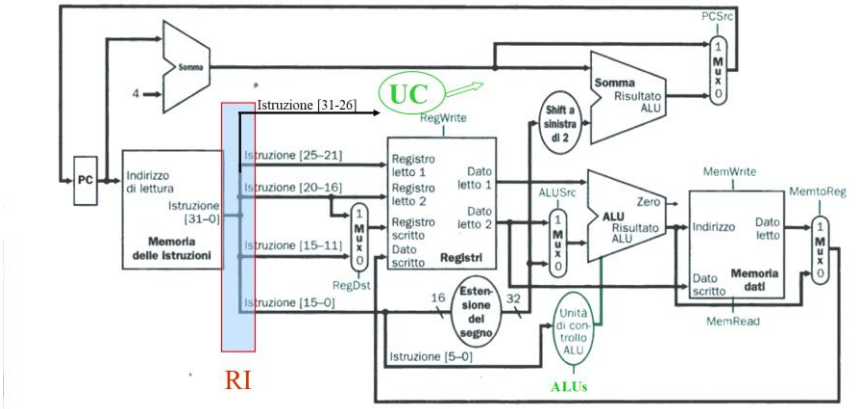
Passo esecuzione	ALU (+4)	Mem Instr.	Register File R	ALU (Pc+Offset)	ALU	Mem Dati	Register File W
Fase fetch							

Fase di Decode



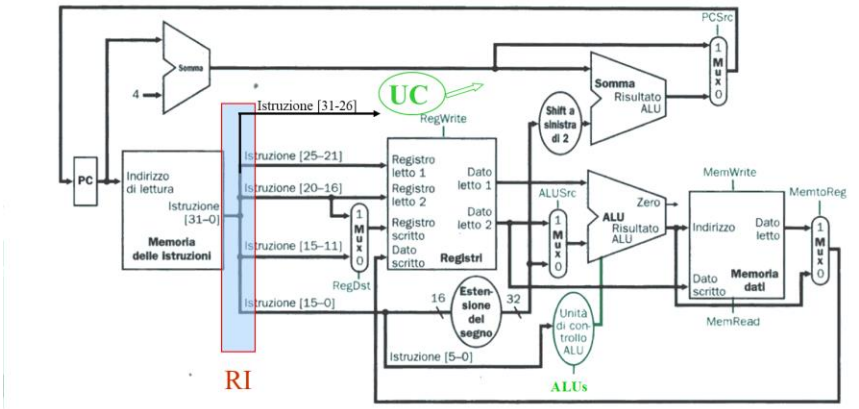
Passo esecuzione	ALU (+4)	Mem Instr.	Register File R	ALU (Pc+Offset)	ALU	Mem Dati	Register File W
decodifica							

Fase di Exe - beq



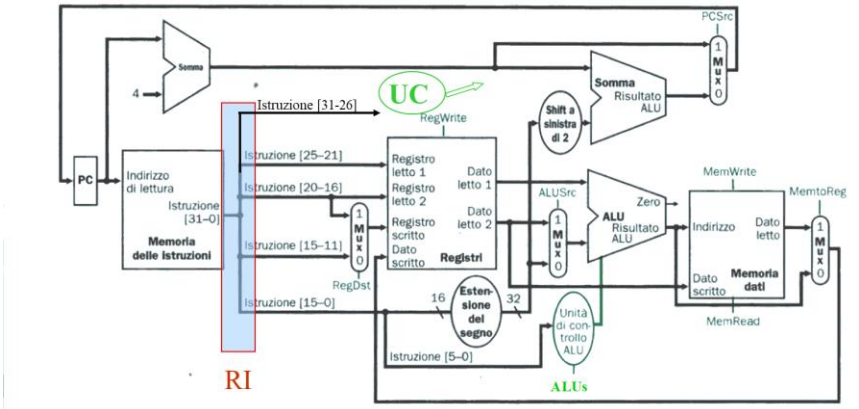
Passo esecuzione	ALU (+4)	Mem Instr.	Register File R	ALU (Pc+Offset)	ALU	Mem Dati	Register File W
decodifica							

Fase di Exe - R



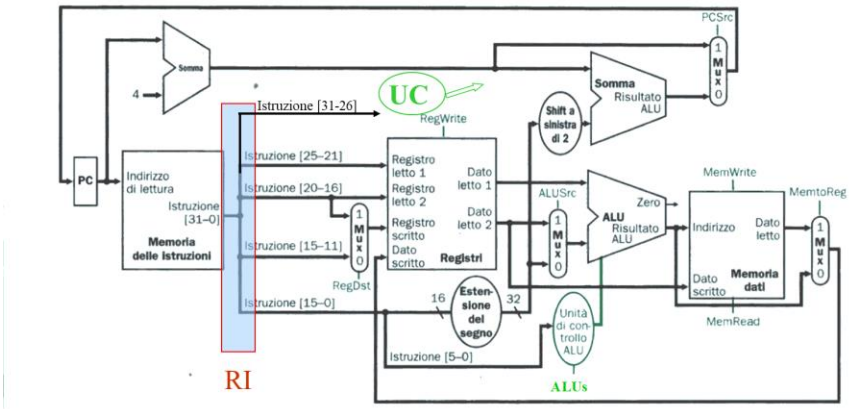
Passo esecuzione	ALU (+4)	Mem Instr.	Register File R	ALU (Pc+Offset)	ALU	Mem Dati	Register File W
decodifica							

Fase di Exe – lw/sw



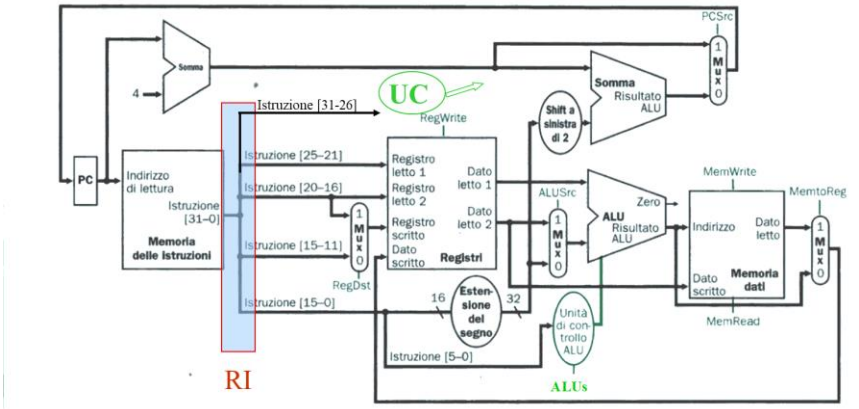
Passo esecuzione	ALU (+4)	Mem Instr.	Register File R	ALU (Pc+Offset)	ALU	Mem Dati	Register File W
decodifica							

Fase di Mem – lw/sw



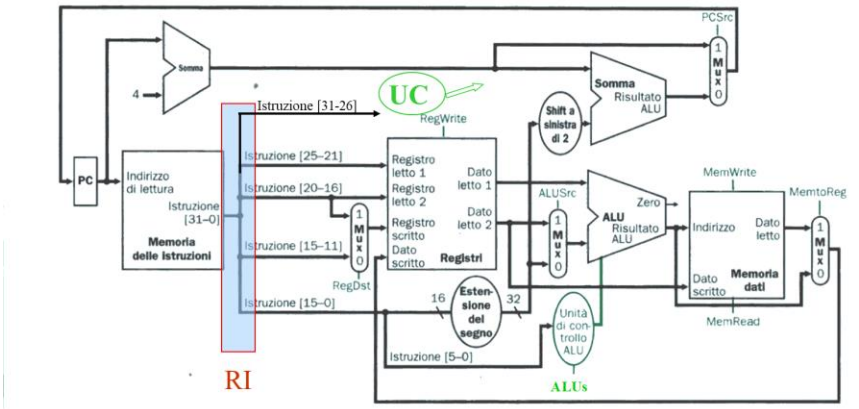
Passo esecuzione	ALU (+4)	Mem Instr.	Register File R	ALU (Pc+Offset)	ALU	Mem Dati	Register File W
decodifica							

Fase di Mem – altre



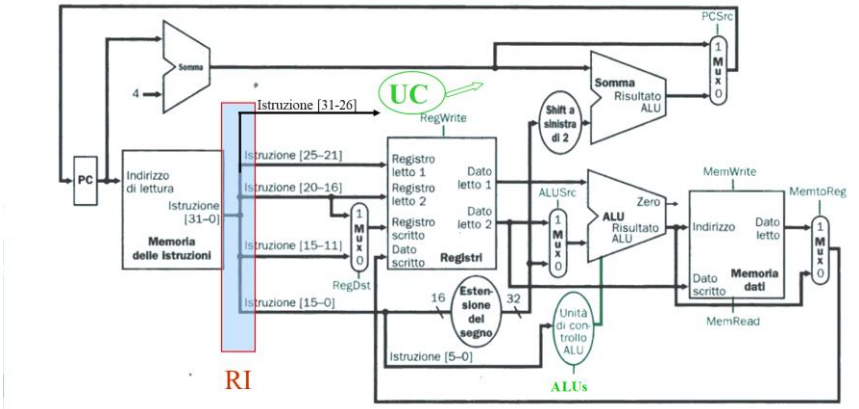
Passo esecuzione	ALU (+4)	Mem Instr.	Register File R	ALU (Pc+Offset)	ALU	Mem Dati	Register File W
decodifica							

Fase di WB – R



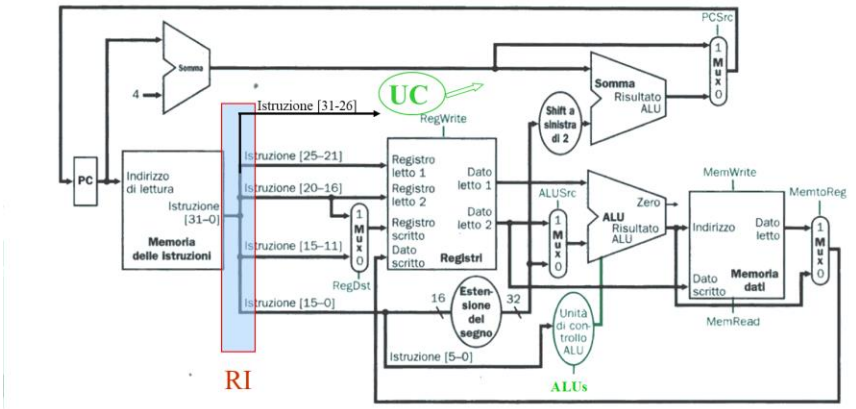
Passo esecuzione	ALU (+4)	Mem Instr.	Register File R	ALU (Pc+Offset)	ALU	Mem Dati	Register File W
decodifica							

Fase di WB – lw



Passo esecuzione	ALU (+4)	Mem Instr.	Register File R	ALU (Pc+Offset)	ALU	Mem Dati	Register File W
decodifica							

Fase di WB – sw/beq



Passo esecuzione	ALU (+4)	Mem Instr.	Register File R	ALU (Pc+Offset)	ALU	Mem Dati	Register File W
decodifica							

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

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 in 1 stadio (al massimo 1/5 del tempo).

⇩

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.

Sommario

I problemi di un'implementazione a singolo ciclo

La pipeline

L'unità di controllo della 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)

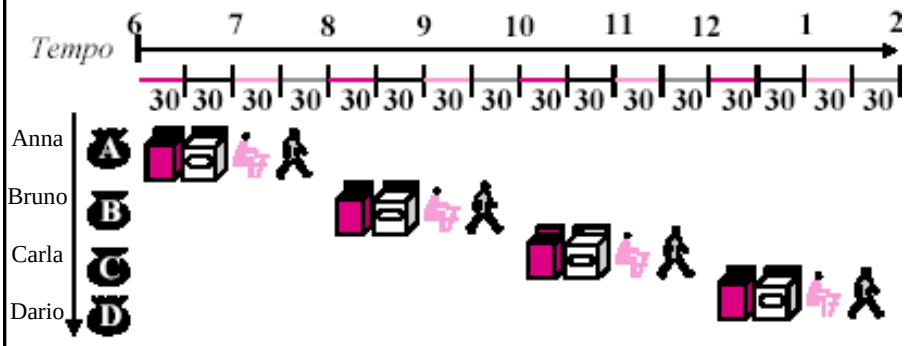
La lavatrice richiede 30 minuti.

L'asciugatrice richiede 30 minuti.

Stirare richiede 30 minuti.

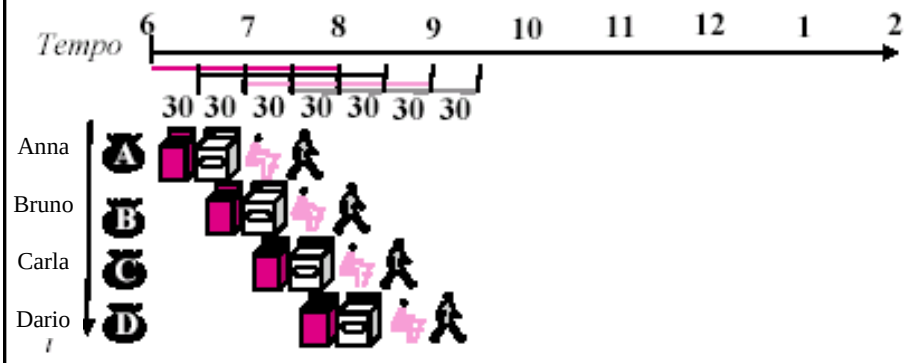
Piegare e mettere via richiede 30 minuti.

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.

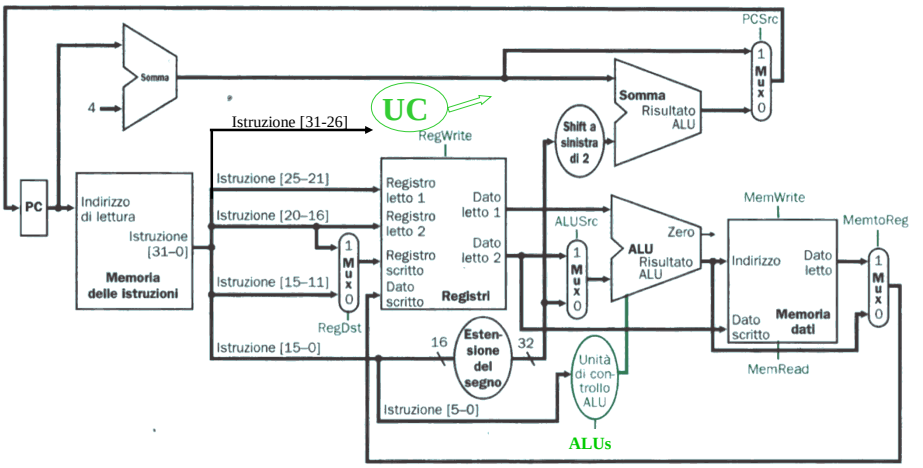
Gli stadi della pipe-line lavorano in contemporanea perché utilizzano unità funzionali differenti.

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

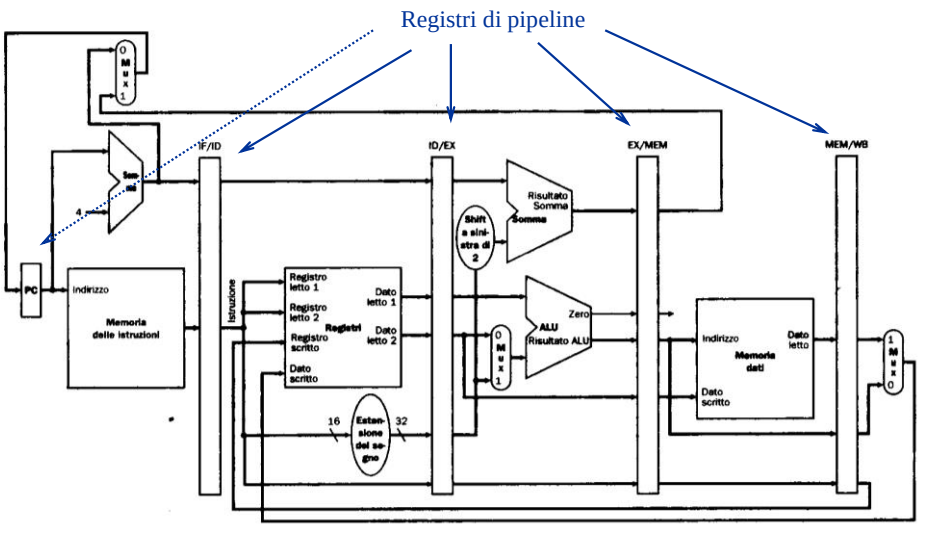
Tutto il materiale che serve per il passo successivo viene prelevato dal passo precedente («ogni lasciata è persa»).

Viene aumentato il throughput.

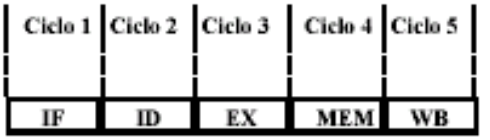
Dalla CPU a singolo ciclo alla CPU con pipeline (il datapath)



CPU con pipeline



I 5 stadi della pipeline



IF/ID ID/EX EX/MEM MEM/WB

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 commuta e trasferisce in uscita al registro a monte dello stadio analizzato l'elaborazione effettuata dallo stadio precente e **tutto quello che serve per l'elaborazione nello stadio corrente e negli stadi successivi.**

Il ruolo dei registri di pipeline

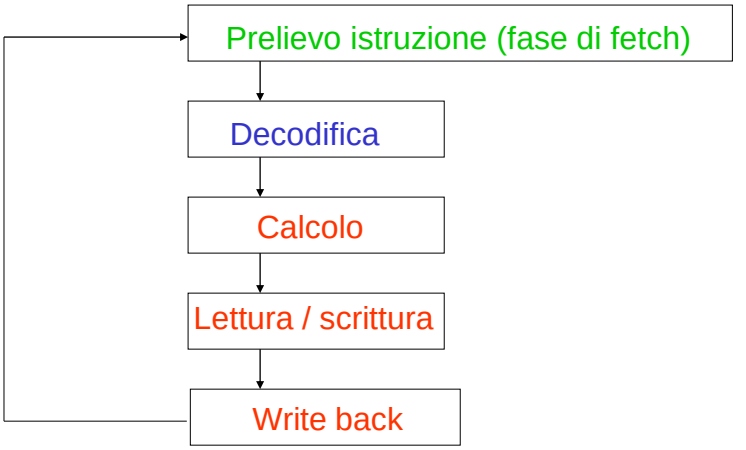
E' un meccanismo assimilabile ad un sistema di chiuse progressive.

Ciascuno stadio produce un risultato. Questo e gli altri dati che servono all'esecuzione degli stadi successivi devono essere memorizzata in un registro.

Il registro mantiene **tutte le informazioni** che servono per l'esecuzione 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.

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

Ciclo di esecuzione di un'istruzione MIPS



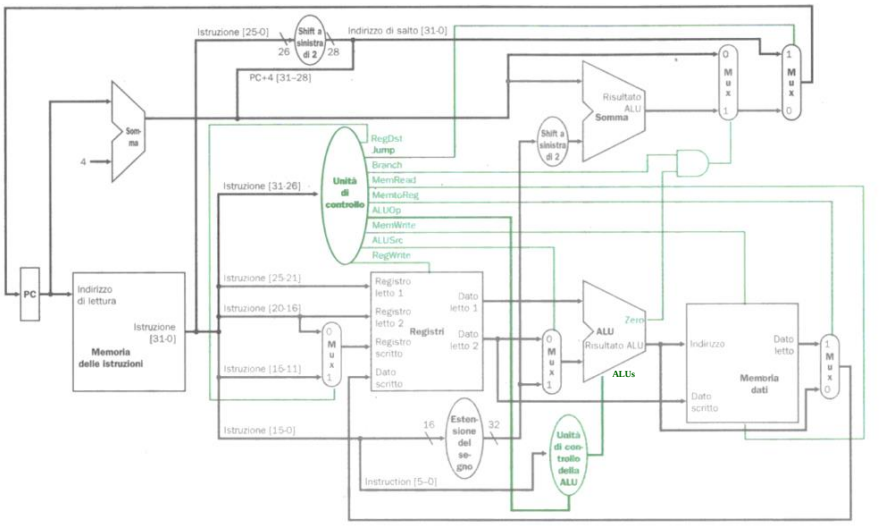
Le istruzioni richiedono 5 passi → 5 stadi della pipe-line

La UC della CPU con pipeline

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

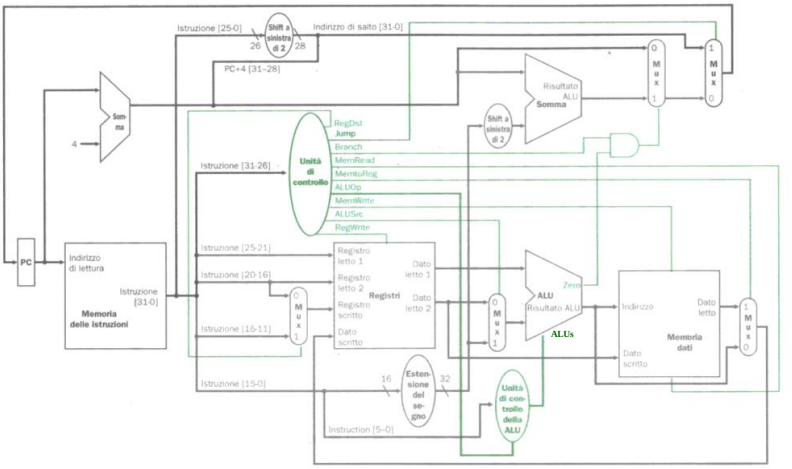
Definizione dell'UC in grado di generare correttamente questi segnali.

I segnali di controllo



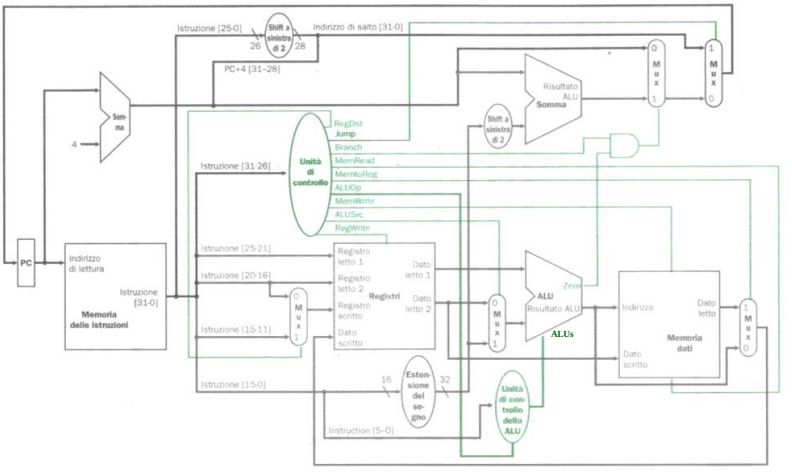
Vengono generati in decodifica ma consumati successivamente

I segnali di controllo - ALUSrc



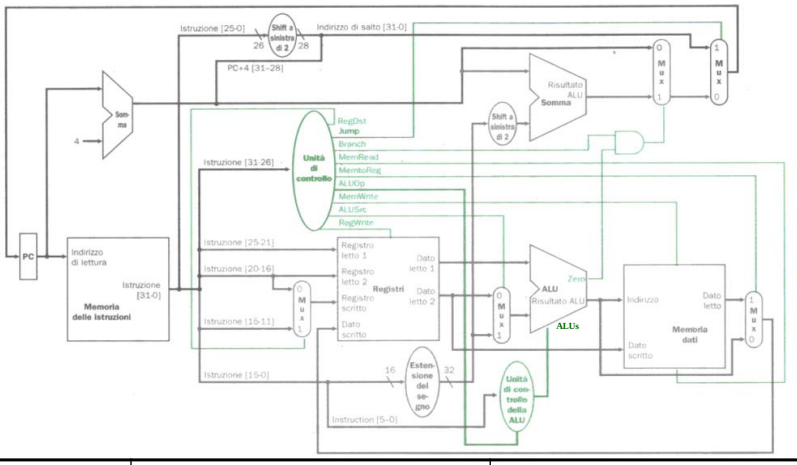
Nome segnale	Effetto quando è negato	Effetto quando è affermato
ALUSrc	Il secondo operando della ALU proviene dalla **seconda porta di lettura** del Register File	Il secondo operando della ALU è la versione estesa (con segno) del campo **costante.**

I segnali di controllo - Branch



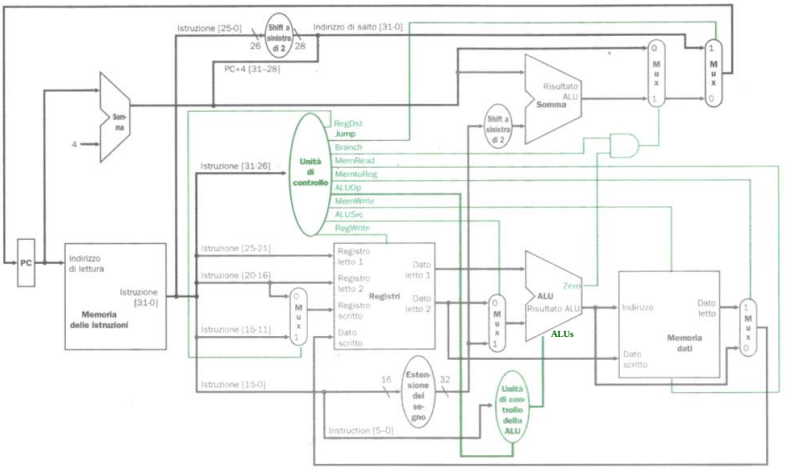
Nome segnale	Effetto quando è negato	Effetto quando è affermato
Branch	Il valore del PC viene sostituito dall'uscita del sommatore che calcola PC+4: **PC = PC + 4**	Il valore del PC viene sostituito dall'uscita del sommatore che calcola la destinazione del salto (**se Zero = 1**): PC = IndirizzoSalto

I segnali di controllo – MR/MW



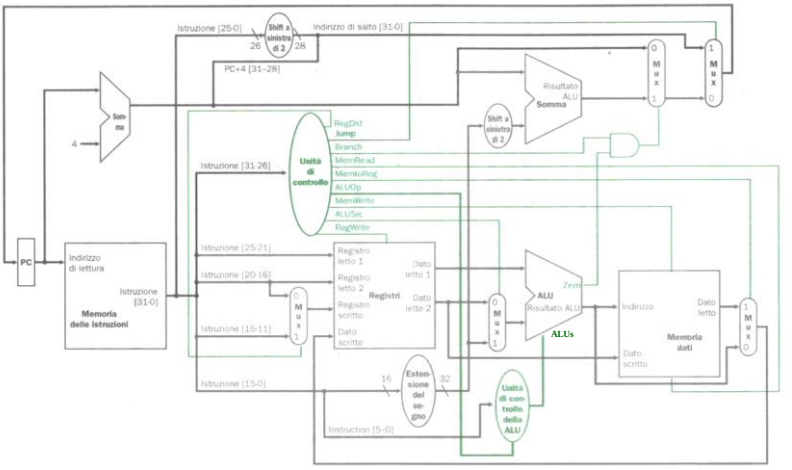
Nome segnale	Effetto quando è negato	Effetto quando è affermato
MemRead	Nessuno	Il contenuto della memoria principale all'indirizzo del MAR **viene letto.**
MemWrite	Nessuno	Il contenuto del MDR **viene scritto nella memoria principale** all'indirizzo nel MAR

I segnali di controllo - RegDst



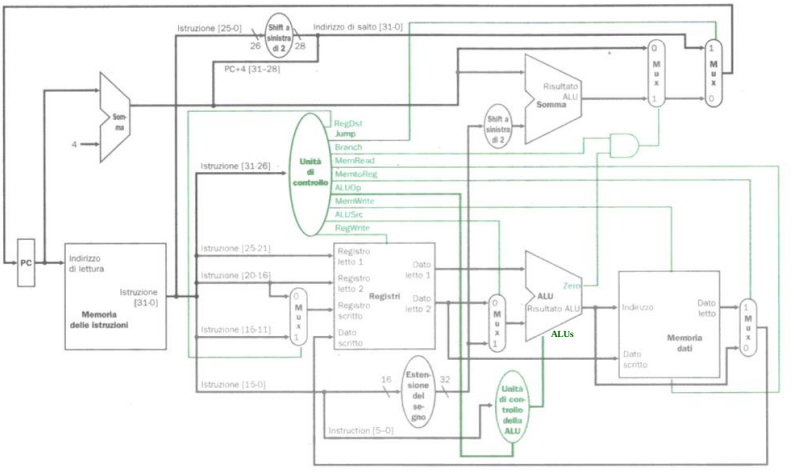
Nome segnale	Effetto quando è negato	Effetto quando è affermato
RegDst	Il numero del registro destinazione proviene dal campo **rt** (R2, bit 20-16)	Il numero del registro destinazione proviene dal campo **rd** (bit 15-11)

I segnali di controllo – Mem2Reg



Nome segnale	Effetto quando è negato	Effetto quando è affermato
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**

I segnali di controllo – RegWrite

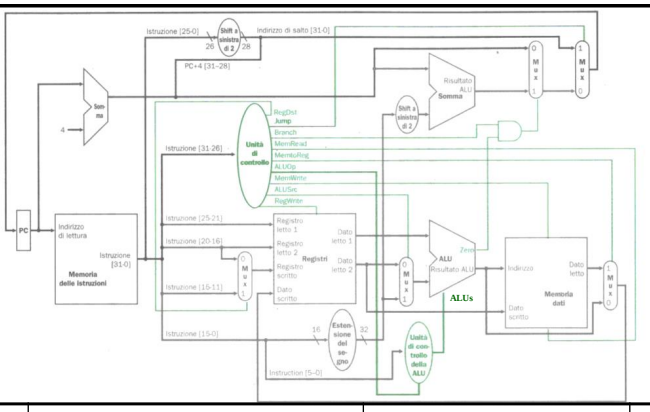


Nome segnale	Effetto quando è negato	Effetto quando è affermato
RegWrite	Nessuno	**Scrittura nel register file** del valore presente all'ingresso Dato Scritto

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).



Raggruppamento per fasi

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

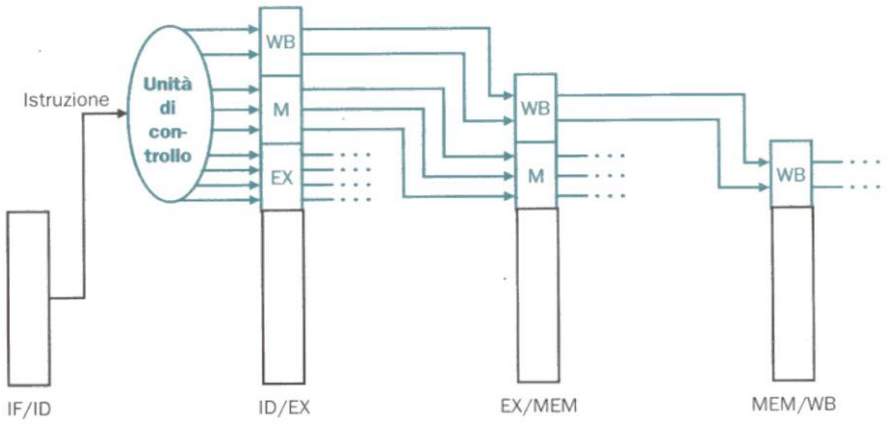
Osservazioni

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	ALUop 1	ALUop 0	ALU Src	Branch	Mem Read	Mem Write	Reg Write	Mem2 Reg
Format-R	1	10 = ‘R format’		0	0	0	0	1	0
lw	0	00 = ‘add’		1	0	1	0	1	1
sw	X	00 = ‘add’		1	0	0	1	0	X
beq	X	01 = ‘sub’		0	1	0	0	0	X

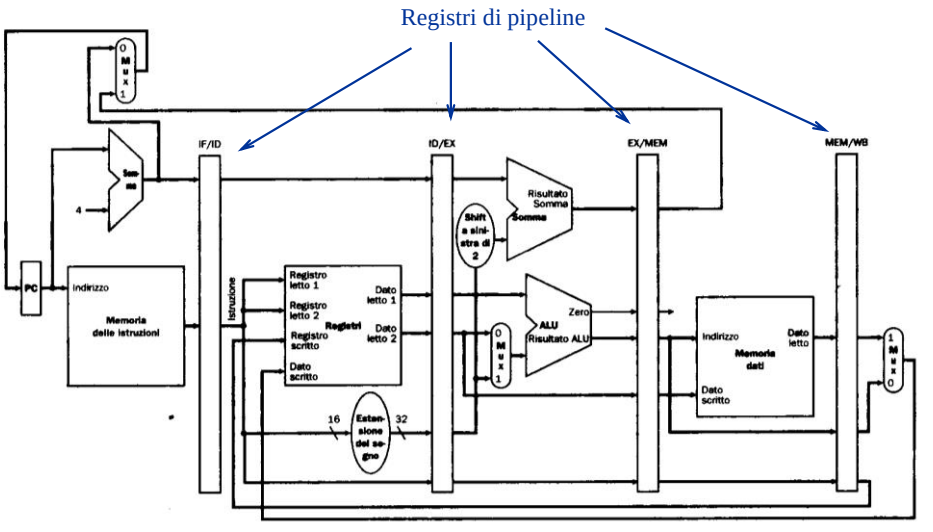
Generazione dei segnali di controllo

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

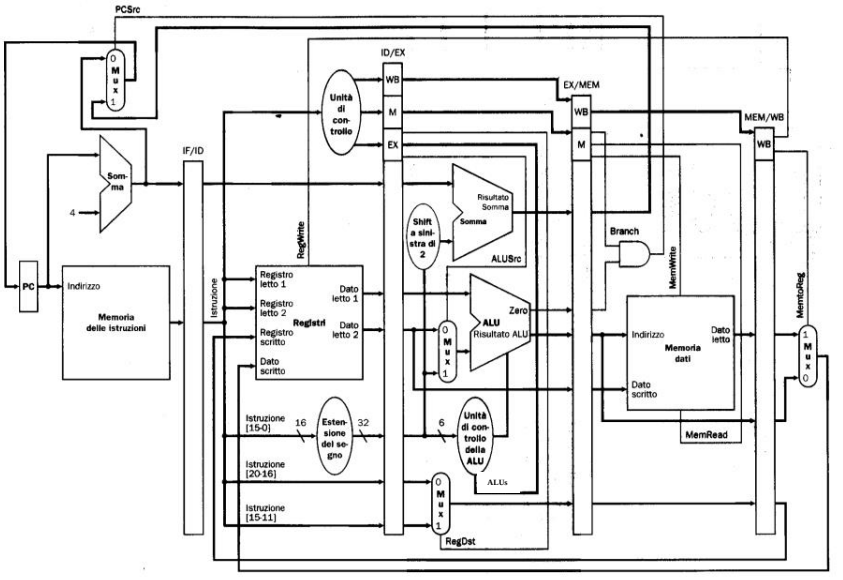


Viene separata la fase di generazione dei segnali di controllo dalla fase di utilizzo.

CPU con pipeline



Pipeline + UC



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.

Miglioramento delle prestazioni

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

Supponiamo un tempo di esecuzione delle istruzioni di 5 ns per una CPU a singolo ciclo con un tempo di esecuzione di ogni fase di 1 ns.

- Il tempo richiesto da una CPU a singolo ciclo per eseguire 100 istruzioni sarà di 500 ns.
- Il tempo di esecuzione richiesto da una CPU con pipeline per eseguire le stesse 100 istruzioni è di è leggermente superiore a 100 ns circa (qualcosa in più perché all'inizio e alla fine del codice la pipeline non sarà piena e per l'introduzione dei registri di pipeline).
- Un'implementazione alternative alla pipeline è la CPU multi-ciclo. In questo caso il clock è dimensionato sulla durata dello stadio di esecuzione e vengono eseguiti solo gli stadi che eseguono lavoro utile (3 stadi per beq, 4 per sw/R, 5 per le lw). Le istuzioni vengono ancora eseguite in sequenza e si può ipotizzare un tempo medio pari a 4 ns. Il tempo di esecuzione richiesto da questa CPU per eseguire le stesse 100 istruzioni sarà di 400 ns.

NB CPU e pipeline a singolo ciclo sono due implementazioni diverse della stessa ISA.

Esempio di esecuzione

Cosa si trova nella pipeline durante l'esecuzione di questo segmento di codice (dati + controllo)?

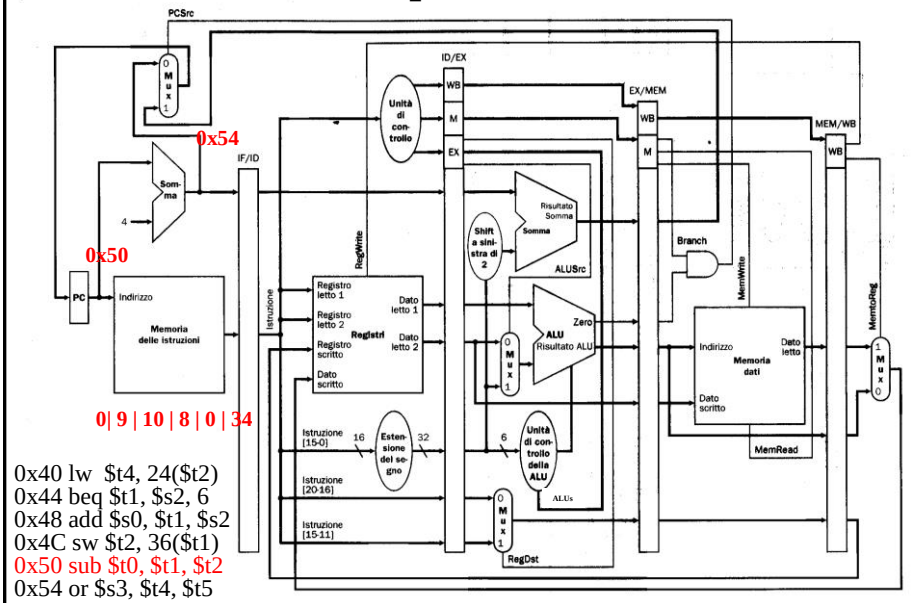
```
0x40 lw  $t4, 24($t2)
0x44 beq $t1, $s2, label     # il valore di label si deriva dall'indirizzo di salto
0x48 add $s0, $t1, $s2
0x4C sw $t2, 36($t1)
0x50 sub $t0, $t1, $t2
0x54 or $s3, $t4, $t5

label: 0x60 ….
```

NB Occorre specificare il contenuto della parte master e slave dei registri di pipeline.

I segnali di controllo fluiscono anch'essi da sinistra verso destra.

Pipeline - IF

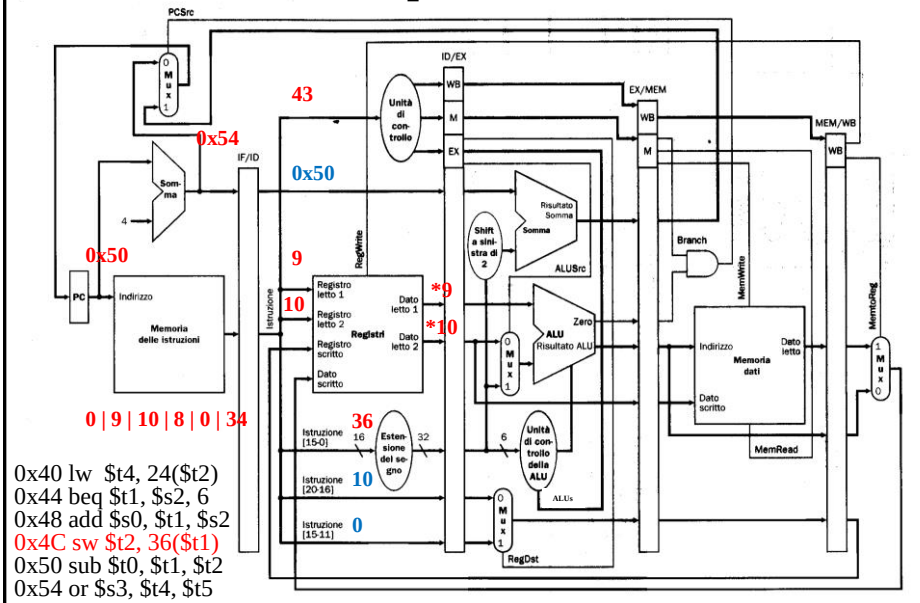
sub



0x40 lw $t4, 24($t2)
0x44 beq $t1, $s2, 6
0x48 add $s0, $t1, $s2
0x4C sw $t2, 36($t1)
0x50 sub $t0, $t1, $t2
0x54 or $s3, $t4, $t5

Pipeline - ID

sub sw



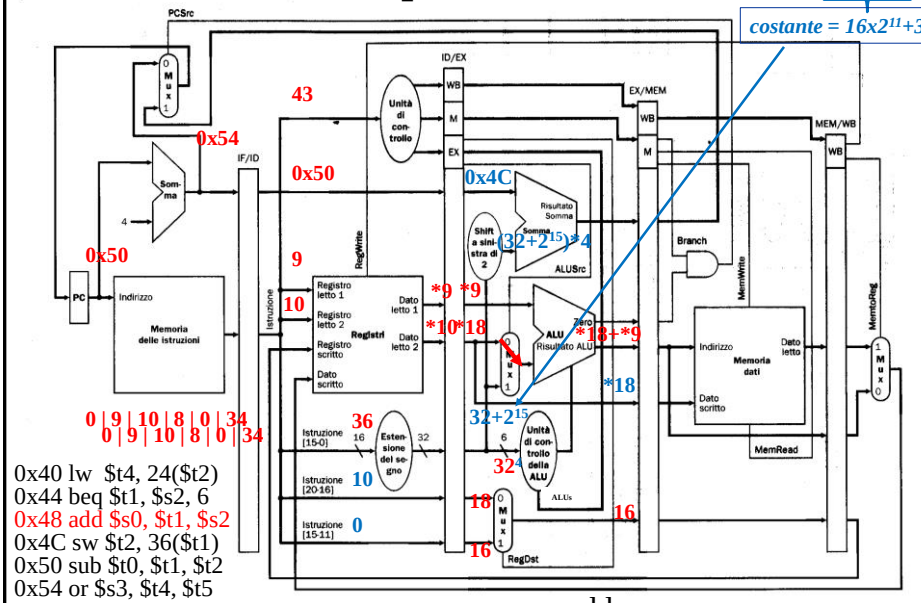
0x40 lw $t4, 24($t2)
0x44 beq $t1, $s2, 6
0x48 add $s0, $t1, $s2
0x4C sw $t2, 36($t1)
0x50 sub $t0, $t1, $t2
0x54 or $s3, $t4, $t5

sub sw

Pipeline - EX

add … = 0 | 9 | 18 | 16 | 0 | 32

costante = $16x2^{11}+32$

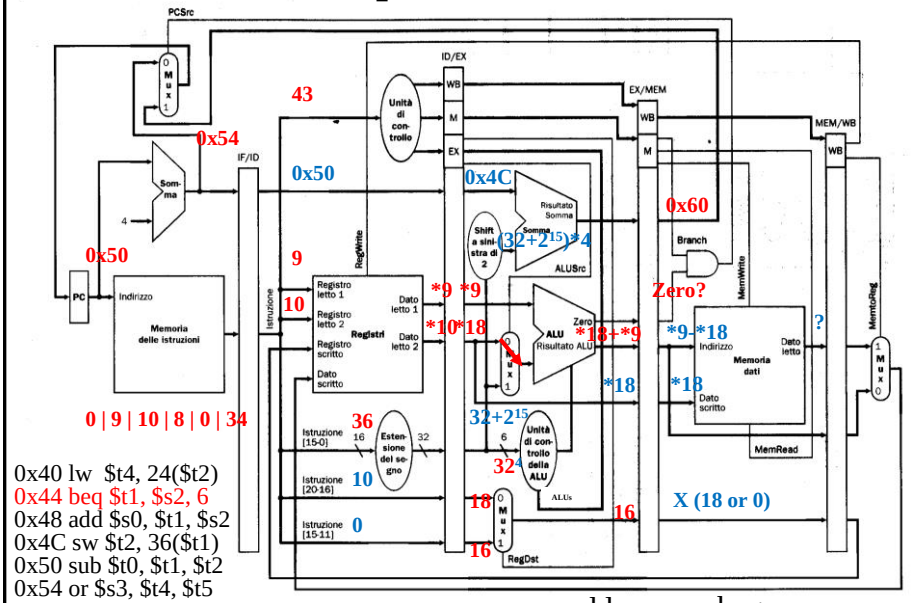


0x40 lw $t4, 24($t2)
0x44 beq $t1, $s2, 6
0x48 add $s0, $t1, $s2
0x4C sw $t2, 36($t1)
0x50 sub $t0, $t1, $t2
0x54 or $s3, $t4, $t5

add

sub sw

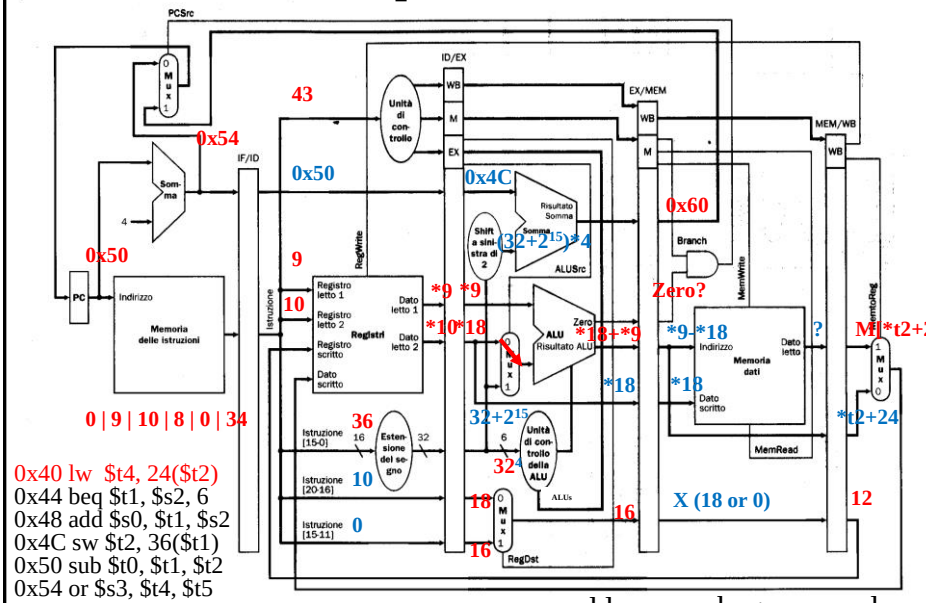
Pipeline - MEM

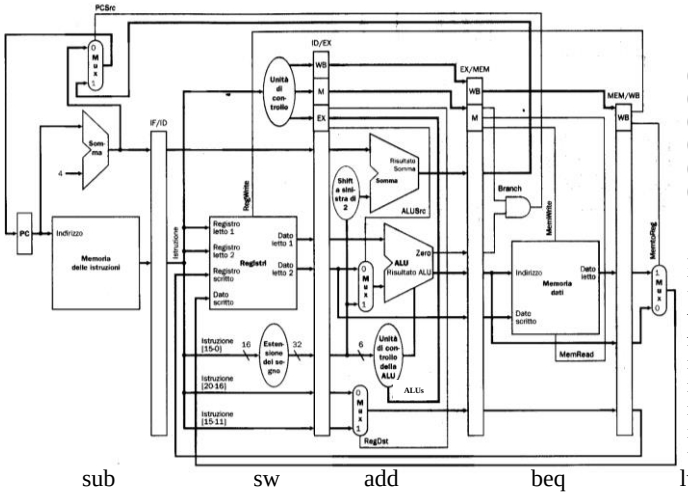


0x40 lw $t4, 24($t2)
0x44 beq $t1, $s2, 6
0x48 add $s0, $t1, $s2
0x4C sw $t2, 36($t1)
0x50 sub $t0, $t1, $t2
0x54 or $s3, $t4, $t5

add beq

sub sw **Pipeline - WB**





0x40 lw $t4, 24($t2)
0x44 beq $t1, $s2, 6
0x48 add $s0, $t1, $s2
0x4C sw $t2, 36($t1)
0x50 sub $t0, $t1, $t2
0x54 or $s3, $t4, $t5

Segnali generate nella fase di decodifica;
RegDst = X
ALUop = 00
ALUSrc = 1
Branch = 0
MemRead = 0
MemWrite = 1
RegWrite = 0
Mem2Reg = X

sub sw add beq lw

Segnali di controllo

Exec (add)				Memory (beq)			WB (lw)	
Reg Dst	ALUop1	ALUop0	ALU Src	Branch	Mem Read	Mem Write	Reg Write	Mem2 Reg
1	10 = "R format"		0	1	0	0	1	1

Alcuni esercizi

Specificare il contenuto di tutti i bus durante l'esecuzione delle seguenti istruzioni:

```
0x00000400 or $s5, $t2, $t1

0x00000404 sw $s1, 1024($s0)

0x00000408 add $t4, $s5, $s1

0x0000040C addi $t1, $t4, 100

0x00000410 beq $s2, $s1, label
```

Si supponga label l'indirizzo 0x000410.

Le istruzioni sono le stesse dell'esecuzione a singolo ciclo di clock. Confrontare le due implementazioni.

Sommario

I problemi di un'implementazione a singolo ciclo

La pipeline

L'unità di controllo della pipeline