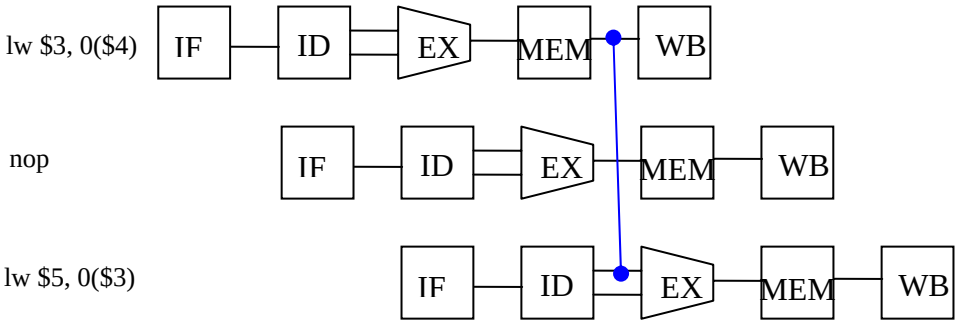
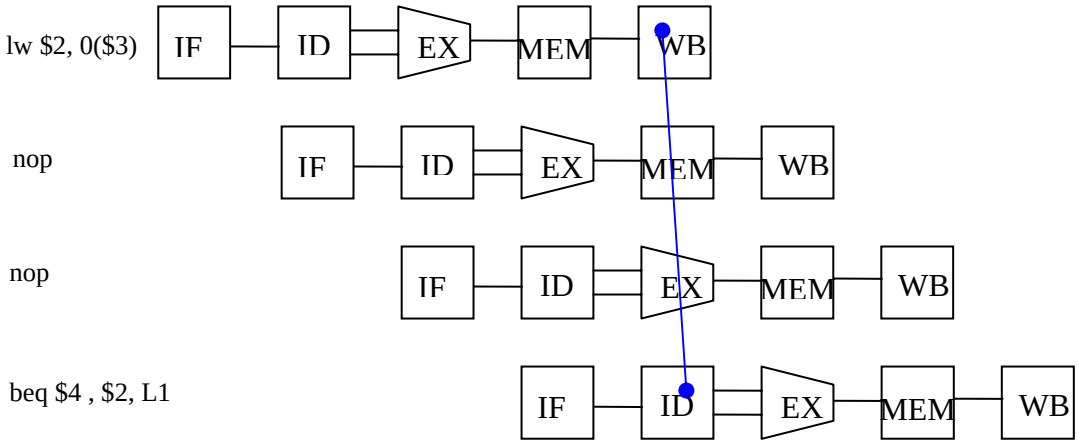
ARCHITETTURE AVANZATE DEI CALCOLATORI, A.A. 2007/08
Soluzione esercizi sul pipelining

Esercizio 1)
N.B. Nei diagrammi a cicli multipli non sono indicati i registri di pipeline (per semplicità)

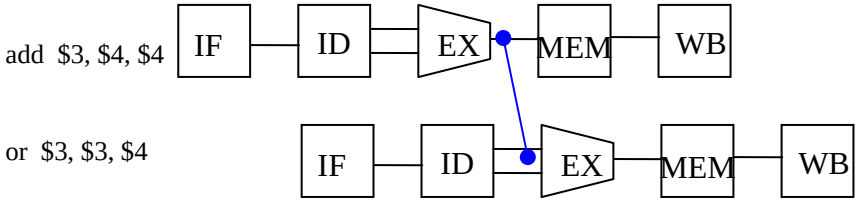
a) Si tratta di una criticità RAW di tipo load-use. Si risolve questa criticità attraverso l'inserimento di 1 **stallo** tra le due istruzioni lw e il **forwarding** del dato letto dalla prima istruzione lw dal registro di pipeline MEM/WB allo stadio EX.



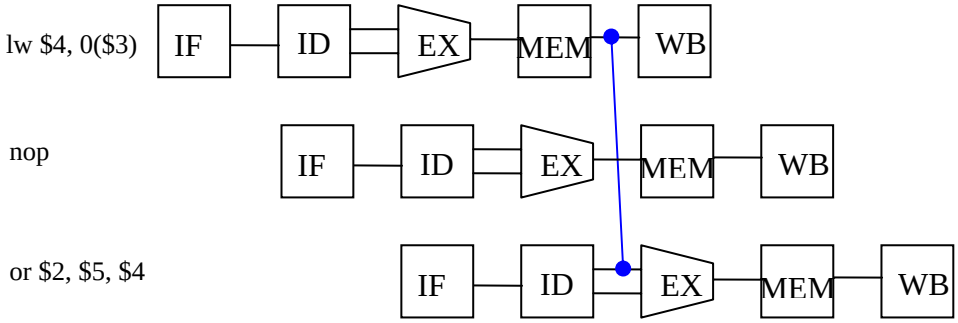
b) Si tratta di una criticità RAW di tipo load-use. Si risolve questa criticità attraverso l'inserimento di 2 **stalli** tra le istruzione lw e beq, l'**anticipo del confronto** al secondo stadio della pipeline (ID) e la lettura e la scrittura del **banco dei registri** nello stesso ciclo di clock (l'istruzione lw scrive $2 nella prima metà del ciclo di clock e l'istruzione beq legge $2 nella seconda metà dello stesso ciclo di clock).



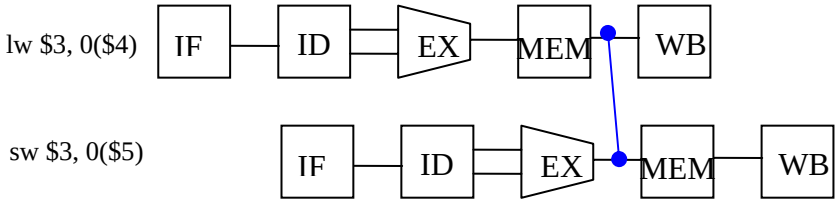
c) Si tratta di una criticità RAW di tipo define-use. Si risolve questa criticità attraverso il **forwarding** del risultato dell'istruzione add dal registro di pipeline EX/MEM allo stadio EX.



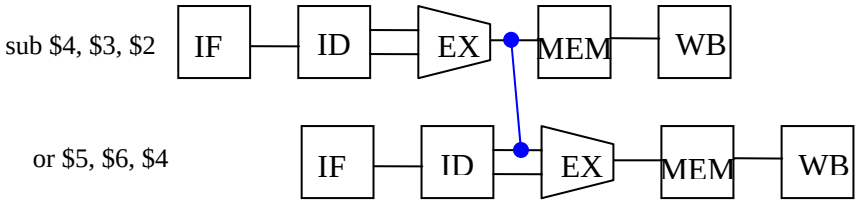
d) Si tratta di una criticità RAW di tipo load-use. Si risolve questa criticità attraverso l'inserimento di 1 **stallo** tra l'istruzione lw e la or e il **forwarding** del dato letto dall'istruzione lw dal registro di pipeline MEM/WB allo stadio EX



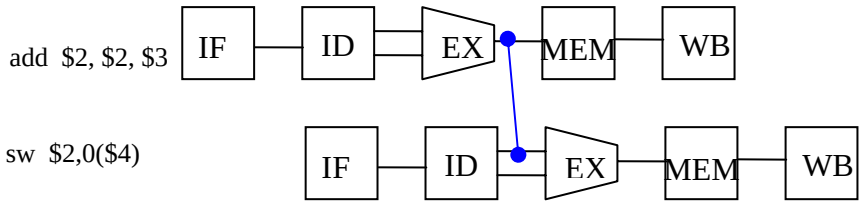
e) Si tratta di una criticità RAW di tipo load-use. Si risolve questa criticità attraverso il **forwarding** del dato letto dall'istruzione lw dal registro di pipeline MEM/WB allo stadio MEM.



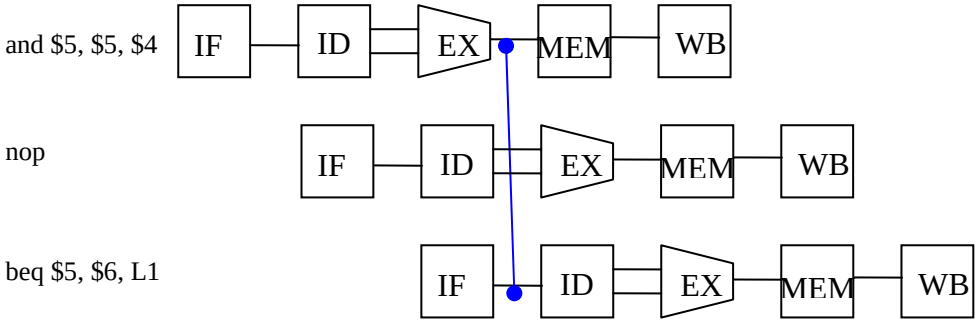
f) Si tratta di una criticità RAW di tipo define-use. Si risolve questa criticità attraverso il **forwarding** del risultato dell'istruzione sub dal registro di pipeline EX/MEM allo stadio EX.



g) Si tratta di una criticità RAW di tipo define-use. Si risolve questa criticità attraverso il **forwarding** del risultato dell'istruzione add dal registro di pipeline EX/MEM allo stadio EX. In alternativa, è possibile effettuare il forwarding del risultato dell'istruzione add dal registro di pipeline MEM/WB allo stadio MEM.



h) Si tratta di una criticità RAW di tipo define-use. Si risolve questa criticità attraverso l'inserimento di 1 **stallo** tra le istruzioni and e beq, il **forwarding** del risultato dell'istruzione and dal registro di pipeline EX/MEM allo stadio ID, in cui è stato **anticipato il confronto**.

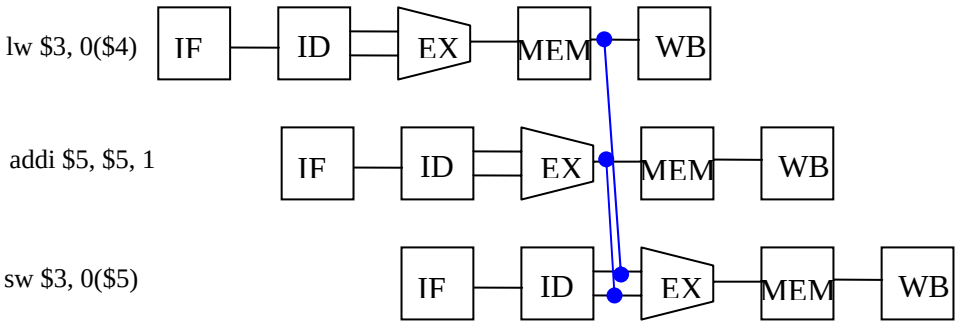


Esercizio 2)

a) Il frammento di codice presenta 2 criticità sui dati di tipo RAW:

1. load-use per il registro $3 tra l'istruzione lw e l'istruzione sw;
2. define-use per il registro $5 tra l'istruzione addi e l'istruzione sw.

La criticità 1 è risolta con il forwarding dal registro di pipeline MEM/WB allo stadio EX; la criticità 2 è risolta con il forwarding dal registro di pipeline EX/MEM allo stadio EX.



Nel caso della criticità 1, per il forwarding è soddisfatta la seguente condizione per il riconoscimento della criticità, vista a lezione e denominata criticità MEM:

if (MEM/WB.RegWrite)
and (MEM/WB.RegisterRd ≠ 0)
and (MEM/WB.RegisterRd = ID/EX.RegisterRt) ForwardB = 01

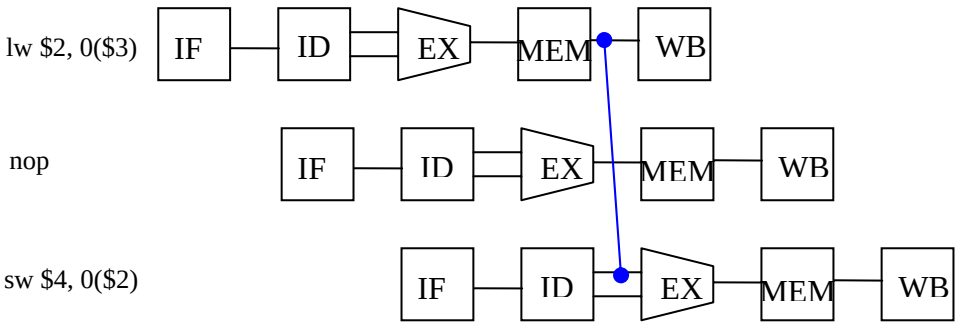
Il primo multiplexer davanti al secondo ingresso della ALU seleziona quindi il valore proveniente dal registro di pipeline MEM/WB; tale valore viene scritto nel registro di pipeline EX/MEM, da dove sarà prelevato al ciclo di clock successivo, in cui avverrà la scrittura della memoria da parte dell'istruzione sw.

Nel caso della criticità 2, per il forwarding è soddisfatta la seguente condizione per il riconoscimento della criticità vista a lezione e denominata criticità EX:

if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd ≠ 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) ForwardA = 10

Il multiplexer davanti al primo ingresso della ALU seleziona quindi il valore proveniente dal registro di pipeline EX/MEM; tale valore viene sommato all'offset ed il risultato della somma è scritto nel registro di pipeline EX/MEM, da dove sarà prelevato al ciclo di clock successivo, in cui avverrà la scrittura della memoria da parte dell'istruzione sw.

b) Il frammento di codice presenta una criticità sui dati di tipo RAW (load-use) per il registro $2 tra l'istruzione lw e l'istruzione sw.
La criticità è risolta con l'inserimento di uno stallo e con il forwarding dal registro di pipeline MEM/WB allo stadio EX.



Nel caso della criticità 1 lo stallo viene inserito con la condizione vista a lezione:

if (ID/EX.MemRead and
and ((ID/EX.RegisterRt = IF/ID.RegisterRs) or
(ID/EX.RegisterRt = IF/ID.RegisterRt)))
stall the pipeline

Per il forwarding, è soddisfatta la seguente condizione per il riconoscimento della criticità vista a lezione e denominata criticità MEM:

if (MEM/WB.RegWrite)
and (MEM/WB.RegisterRd ≠ 0)
and (MEM/WB.RegisterRd = ID/EX.RegisterRs) ForwardA = 01

Il multiplexer davanti al primo ingresso della ALU seleziona quindi il valore proveniente dal registro di pipeline MEM/WB; tale valore viene sommato all'offset ed il risultato della somma è scritto nel registro di pipeline EX/MEM, da dove sarà prelevato al ciclo di clock successivo, in cui avverrà la scrittura della memoria da parte dell'istruzione sw.

Esercizio 3)

Non è necessario utilizzare la tecnica del forwarding in quanto, al momento dell'accesso in memoria da parte dell'istruzione `lw`, il contenuto del registro `$2` è già stato immagazzinato nel ciclo di clock immediatamente precedente dall'istruzione `sw`. Di conseguenza, l'istruzione `lw` accede ad una cella di memoria consistente, non compromettendo l'integrità dei dati del flusso del programma.

Esercizio 4)

a) In totale vengono eseguite 30 istruzioni di salto. Infatti, il programma consiste in due cicli annidati, in cui il ciclo più interno (etichettato da L1) esegue 5 istruzioni di salto. Perciò, per ogni ripetizione del ciclo più esterno (etichettato da L2), saranno eseguite 5 istruzioni di salto da parte del ciclo interno più 1 istruzione di salto del ciclo esterno. In totale vengono eseguite 6 istruzioni di salto per ogni ripetizione del ciclo esterno, che in tutto si ripete 5 volte, da cui 6*5=30.

b) Gli indirizzi delle istruzioni di salto sono 52 e 60 che in base 2 equivalgono rispettivamente a:

$(52)_{10}$ = $(0000\ 0000\ 0000\ 0000\ 0000\ 0011\ 0100)_2$
$(60)_{10}$ = $(0000\ 0000\ 0000\ 0000\ 0000\ 0011\ 1100)_2$

Quindi, entrambe le istruzioni di salto hanno gli ultimi 3 bit meno significativi uguali; di conseguenza, entrambe punteranno alla stessa riga della BHT, la entry 4, che è inizializzata a 1 (T).

La sequenza effettiva dei salti nel codice è:
TTTTN T TTTTN T TTTTN T TTTTN T TTTTN N
in cui si sono indicate in blu le istruzioni di salto del ciclo più interno, in rosso quelle del ciclo esterno.

Poiché la riga della BHT acceduta da entrambe le istruzioni di salto è inizializzata a 1, verrà predetto un T per il primo salto che farà riferimento a detta riga, da cui si ottiene la seguente sequenza di predizioni:
TTTTT N TTTTT N TTTTT N TTTTT N TTTTT N
dove sono state sottolineate le predizioni errate rispetto alla sequenza effettiva dei salti.
In totale sono stati predetti correttamente 21 salti ed erroneamente 9 salti.

c) Accuratezza predizione = 21/30 = 70%

Esercizio 5)
a) Prima dello srotolamento del ciclo, occorrono 9 cicli di clock per iterazione (8 istruzioni ed uno stallo dopo bne dovuto alla criticità sul controllo).

Poiché il ciclo for parte da i=1 e la condizione di uscita è i<3, allora si srotola il loop per 2 volte. Diviene quindi inutile incrementare i registri $s0 e $s1. Anche l'istruzione bne diviene inutile, in quanto il loop è completamente srotolato.

Srotolando il ciclo per 2 volte, il codice MIPS diventa (senza ridenominazione dei registri):

```
lw $t0, 0($s0)
lw $t1, 0($s1)
add $s2, $s2, $t0
add $s2, $s2, $t1
add $s3, $s3, $t0
lw $t0, 4($s0)
lw $t1, 4($s1)
add $s2, $s2, $t0
add $s2, $s2, $t1
add $s3, $s3, $t0
```

Con ridenominazione dei registri:

```
lw $t0, 0($s0)
lw $t1, 0($s1)
add $s2, $s2, $t0
add $s2, $s2, $t1
add $s3, $s3, $t0
lw $t2, 4($s0)
lw $t3, 4($s1)
add $s2, $s2, $t2
add $s2, $s2, $t3
add $s3, $s3, $t2
```

Riordinando il codice:

```
lw $t0, 0($s0)
lw $t1, 0($s1)
lw $t2, 4($s0)
lw $t3, 4($s1)
add $s2, $s2, $t0
add $s2, $s2, $t1
add $s3, $s3, $t0
add $s2, $s2, $t2
add $s2, $s2, $t3
add $s3, $s3, $t2
```

Tutte le criticità di tipo RAW presenti nel codice riordinato sono risolvibili mediante forwarding.
Quindi, CPI = 10/10 = 1 (10 cicli di clock per 2 iterazioni).

b) Effettuando lo scheduling su MIPS con pipeline statica 2-issue (prima istruzione: ALU o branch, seconda istruzione: load o store):

Istruzione ALU o branch	**Istruzione load o store**	**Ciclo di clock**
`nop`	`lw $t0, 0($s0)`	1
`nop`	`lw $t1, 0($s1)`	2
`add $s2, $s2, $t0`	`lw $t2, 4($s0)`	3
`add $s2, $s2, $t1`	`lw $t3, 4($s1)`	4
`add $s3, $s3, $t0`		5
`add $s2, $s2, $t2`		6
`add $s2, $s2, $t3`		7
`add $s3, $s3, $t2`		8

Quindi, CPI = 8/10 = 0.8 (8 cicli di clock per 2 iterazioni).

Esercizio 6)
a) Si individuano le criticità sui dati (in rosso) e sul controllo (in blu) presenti nel codice.

Istruzione	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13
`L1: lw $2, BASEA($4)`	IF	ID	EX	MEM	WB								
`addi $2, $2, INC1`		IF	ID	EX	MEM	WB							
`lw $3, BASEB($4)`			IF	ID	EX	MEM	WB						
`addi $3, $3, INC2`				IF	ID	EX	MEM	WB					
`add $5, $2, $3`					IF	ID	EX	MEM	WB				
`sw $5, BASEC($4)`						IF	ID	EX	MEM	WB			
`addi $4, $4, 4`							IF	ID	EX	MEM	WB		
`bne $4, $7, L1`								IF	ID	EX	MEM	WB	
									IF	ID	EX	MEM	WB

Si indicano nella seconda colonna il numero di stalli da inserire prima di ciascuna istruzione in modo da risolvere le criticità presenti e nella terza colonna il tipo di criticità.

Istruzione	Numero stalli	Tipo di criticità
`L1: lw $2, BASEA($4)`		
`addi $2, $2, INC1`	3	D (load/use)
`lw $3, BASEB($4)`		
`addi $3, $3, INC2`	3	D (load/use)
`add $5, $2, $3`	3	D (define/use)
`sw $5, BASEC($4)`	3	D (define/use)
`addi $4, $4, 4`		
`bne $4, $7, L1`	3	D (define/use)
	3	C

Numero di stalli = 18
$CPI_{asintotico} = (8 + 18)/8 = 26/8 = 3{,}25$

b) Si individuano le criticità sui dati (in rosso) e sul controllo (in blu) che non possono essere risolte solo mediante forwarding e si indicano (in verde) i percorsi di forwarding usati e (in arancione) la lettura/scrittura del banco dei registri.

Istruzione	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13
`L1: lw $2, BASEA($4)`	IF	ID	EX	MEM	WB								
`addi $2, $2, INC1`		IF	ID	EX	MEM	WB							
`lw $3, BASEB($4)`			IF	ID	EX	MEM	WB						
`addi $3, $3, INC2`				IF	ID	EX	MEM	WB					
`add $5, $2, $3`					IF	ID	EX	MEM	WB				
`sw $5, BASEC($4)`						IF	ID	EX	MEM	WB			
`addi $4, $4, 4`							IF	ID	EX	MEM	WB		
`bne $4, $7, L1`								IF	ID	EX	MEM	WB	
									IF	ID	EX	MEM	WB

Nota: la freccia arancione in realtà non è all'indietro ma in verticale, dal momento che l'istruzione add $5, $2, $3 è ritardata di un ciclo rispetto a quanto mostrato in tabella.

Si indicano nella seconda colonna il numero di stalli da inserire prima di ciascuna istruzione in modo da risolvere le criticità presenti, nella terza colonna il tipo di criticità e nella quarta colonna il percorso di forwarding usato.

Istruzione	Numero stalli	Tipo di criticità	Percorso di forwarding
`L1: lw $2, BASEA($4)`			
`addi $2, $2, INC1`	1	D	MEM/WB-EX
`lw $3, BASEB($4)`			
`addi $3, $3, INC2`	1	D	MEM/WB-EX
`add $5, $2, $3`		D	EX/MEM-EX
`sw $5, BASEC($4)`		D	EX/MEM-EX
`addi $4, $4, 4`			
`bne $4, $7, L1`	1	D	EX/MEM-ID
	1	C	

Numero di stalli = 4
$CPI_{asintotico} = (8 + 4)/8 = 12/8 = 1,5$

Esercizio 7)
a) Nel MIPS non è noto l'indirizzo di salto prima del risultato del confronto. Non c'è vantaggio nell'adottare un approccio di tipo Branch Always Taken. I risultati sono identici a quelli dell'esercizio 6 punto (b).

b) Riordinando il codice nel modo seguente e sfruttando il branch delay slot è possibile eliminare completamente gli stalli ed ottenere un CPI a 1.

```
L1:   lw $2, BASEA($4)
      lw $3, BASEB($4)
      addi $4, $4, 4
      addi $2, $2, INC1
```

```
addi $3, $3, INC2
add $5, $2, $3
bne $4, $7, L1
sw $5, BASEC-4($4)
```

Istruzione	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13
L1: lw $2, BASEA($4)	IF	ID	EX	MEM	WB								
lw $3, BASEB($4)		IF	ID	EX	MEM	WB							
addi $4, $4, 4			IF	ID	EX	MEM	WB						
addi $2, $2, INC1				IF	ID	EX	MEM	WB					
addi $3, $3, INC2					IF	ID	EX	MEM	WB				
add $5, $2, $3						IF	ID	EX	MEM	WB			
bne $4, $7, L1							IF	ID	EX	MEM	WB		
sw $5, BASEC-4($4)								IF	ID	EX	MEM	WB	
									IF	ID	EX	MEM	WB

Si indicano nella seconda colonna il numero di stalli da inserire prima di ciascuna istruzione in modo da risolvere le criticità rimaste, nella terza colonna il tipo di criticità e nella quarta colonna il percorso di forwarding usato.

Istruzione	Numero stalli	Tipo di criticità	Percorso di forwarding
L1: lw $2, BASEA($4)			
lw $3, BASEB($4)			
addi $4, $4, 4			
addi $2, $2, INC1			
addi $3, $3, INC2			
add $5, $2, $3		D	MEM/WB-EX , EX/MEM-EX
bne $4, $7, L1			
sw $5, BASEC-4($4)		D	MEM/WB-EX

Numero di stalli = 0

$CPI_{asintotico} = 8/8 = 1$

Uno scheduling alternativo che riduce ugualmente il CPI ad 1 è il seguente:

```
L1:   lw $2, BASEA($4)
      lw $3, BASEB($4)
      addi $2, $2, INC1
      addi $3, $3, INC2
      addi $4, $4, 4
      add $5, $2, $3
      bne $4, $7 ,L1
      sw $5, BASEC-4($4)
```

Esercizio 8)
a) Il ciclo viene iterato per 2 volte.
Si analizza una iterazione del ciclo, evidenziando le criticità sui dati (in rosso) e sul controllo (in blu) presenti nel codice.

Istruzione	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13
`L1: lw $t0, 0($s0)`	IF	ID	EX	MEM	WB								
`lw $t1, 4($s0)`		IF	ID	EX	MEM	WB							
`add $t2, $t0, $t1`			IF	ID	EX	MEM	WB						
`sw $t1, 0($s0)`				IF	ID	EX	MEM	WB					
`sw $t2, 4($s0)`					IF	ID	EX	MEM	WB				
`addi $s1, $s1, -1`						IF	ID	EX	MEM	WB			
`Bne $s1, $0, L1`							IF	ID	EX	MEM	WB		
								IF	ID	EX	MEM	WB	

Si indicano nella seconda colonna il numero di stalli da inserire prima di ciascuna istruzione in modo da risolvere le criticità presenti e nella terza colonna il tipo di criticità.

Istruzione	Numero stalli	Tipo di criticità
`L1: lw $t0, 0($s0)`		
`lw $t1, 4($s0)`		
`add $t2, $t0, $t1`	3	D (load/use)
`sw $t1, 0($s0)`		D (load/use)
`sw $t2, 4($s0)`	2	D (define/use)
`addi $s1, $s1, -1`		
`bne $s1, $0, L1`	3	D (define/use)
	3	C

Numero di stalli per iterazione = 11
Numero di stalli per intero ciclo = 11*2 = 22
Numero di cicli di clock per intero ciclo = 7*2 + 11*2 = 36

b) Si individuano le criticità sui dati (in rosso) e sul controllo (in blu) che non possono essere risolte solo mediante forwarding e si indicano (in verde) i percorsi di forwarding usati e (in arancione) la lettura/scrittura del banco dei registri.

Istruzione	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13
`L1: lw $t0, 0($s0)`	IF	ID	EX	MEM	WB								
`lw $t1, 4($s0)`		IF	ID	EX	MEM	WB							
`add $t2, $t0, $t1`			IF	ID	EX	MEM	WB						
`sw $t1, 0($s0)`				IF	ID	EX	MEM	WB					
`sw $t2, 4($s0)`					IF	ID	EX	MEM	WB				
`addi $s1, $s1, -1`						IF	ID	EX	MEM	WB			
`bne $s1, $0, L1`							IF	ID	EX	MEM	WB		
								IF	ID	EX	MEM	WB	

Nota: la freccia arancione in realtà non è all'indietro ma in verticale, dal momento che le istruzioni add $t2, $t0, $t1 e sw $t12, 0($s0) sono ritardate di un ciclo rispetto a quanto mostrato in tabella.

Si indicano nella seconda colonna il numero di stalli da inserire prima di ciascuna istruzione in modo da risolvere le criticità presenti, nella terza colonna il tipo di criticità e nella quarta colonna il percorso di forwarding usato (indicando il registro di pipeline da cui si propaga).

Istruzione	Numero stalli	Tipo di criticità	Percorso di forwarding
`L1: lw $t0, 0($s0)`			
`lw $t1, 4($s0)`			
`add $t2, $t0, $t1`	1	D	MEM/WB-EX
`sw $t1, 0($s0)`		D	
`sw $t2, 4($s0)`		D	MEM/WB-EX
`addi $s1, $s1, -1`			
`bne $s1, $0, L1`	1	D	EX/MEM-ID
	1	C	

Numero di stalli per iterazione = 3
Numero di stalli per intero ciclo = 3*2 = 6
Numero di cicli di clock per intero ciclo = 7*2 + 3*2 = 20

c) E' possibile riordinare il codice in modo tale da eliminare gli stalli rimasti al punto b) nel modo seguente:

```
L1: lw $t0, 0($s0)
   lw $t1, 4($s0)
   addi $s1, $s1, -1
   add $t2, $t0, $t1
   sw $t1, 0($s0)
   bne $s1, $0, L1
   sw $t2, 4($s0)
```

Si indicano (in verde) i percorsi di forwarding usati e (in arancione) la lettura/scrittura del banco dei registri.

Istruzione	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13
`L1: lw $t0, 0($s0)`	IF	ID	EX	MEM	WB								
`lw $t1, 4($s0)`		IF	ID	EX	MEM	WB							
`addi $s1, $s1, -1`			IF	ID	EX	MEM	WB						
`add $t2, $t0, $t1`				IF	ID	EX	MEM	WB					
`sw $t1, 0($s0)`					IF	ID	EX	MEM	WB				
`bne $s1, $0, L1`						IF	ID	EX	MEM	WB			
`sw $t2, 4($s0)`							IF	ID	EX	MEM	WB		
								IF	ID	EX	MEM	WB	

Numero di stalli per iterazione = 0
Numero di stalli per intero ciclo = 3*2 = 6
Numero di cicli di clock per intero ciclo = 7*2 = 14