Sia dato il seguente ciclo di un programma in linguaggio macchina MIPS. Si supponga che i registri $t6, e $t7 siano stati inizializzati rispettivamente ai valori 0 e 4N. I simboli BASEA, BASEB, e BASEC sono costanti a 16 bit, prefissate.
Si consideri una singola iterazione del ciclo eseguita dal processore MIPS in modalità pipeline a 5 stadi senza ottimizzazioni.
Individuare i conflitti sui dati di tipo RAW (Read After Write) e i conflitti sul controllo presenti nel programma e indicarli nell'ultima colonna. Inserire nella prima colonna il numero di stalli da inserire prima di ciascuna istruzione in modo da risolvere i conflitti presenti nel programma.

n. stalli	TIPO CONFL.	ISTRUZIONE	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C15
		L1: beq $t6,$t7, EXIT	IF	ID	EX	ME	WB											
		lw $t2,BASEA($t6)		IF	ID	EX	ME	WB										
		add $t2,$t2,$t2			IF	ID	EX	ME	WB									
		sw $t2,BASEA($t6)				IF	ID	EX	ME	WB								
		lw $t3,BASEB($t6)					IF	ID	EX	ME	WB							
		add $t3,$t3,$t3						IF	ID	EX	ME	WB						
		sw $t3,BASEB($t6)							IF	ID	EX	ME	WB					
		add $t4,$t2,$t3								IF	ID	EX	ME	WB				
		sw $t4,BASEC($t6)									IF	ID	EX	ME	WB			
		addi $t6,$t6,4										IF	ID	EX	ME	WB		
		J L1											IF	ID	EX	ME	WB	
		EXIT:												IF	ID	EX	ME	WB

n. stalli	TIPO CONFL.	ISTRUZIONE	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C15
		L1: beq $t6,$t7, EXIT	IF	ID	EX	ME	WB											
3	C	lw **$t2**,BASEA($t6)		IF	ID	EX	ME	WB										
3	D	add **$t2,$t2**,$t2			IF	ID	EX	ME	WB									
3	D	sw **$t2**,BASEA($t6)				IF	ID	EX	ME	WB								
		lw **$t3**,BASEB($t6)					IF	ID	EX	ME	WB							
3	D	add $t3**,$t3,$t3**						IF	ID	EX	ME	WB						
3	D	sw $t3,BASEB($t6)							IF	ID	EX	ME	WB					
		add $t4,$t2,$t3								IF	ID	EX	ME	WB				
3	D	sw $t4,BASEC($t6)									IF	ID	EX	ME	WB			
		addi $t6,$t6,4										IF	ID	EX	ME	WB		
		J L1											IF	ID	EX	ME	WB	
3	C	EXIT:												IF	ID	EX	ME	WB

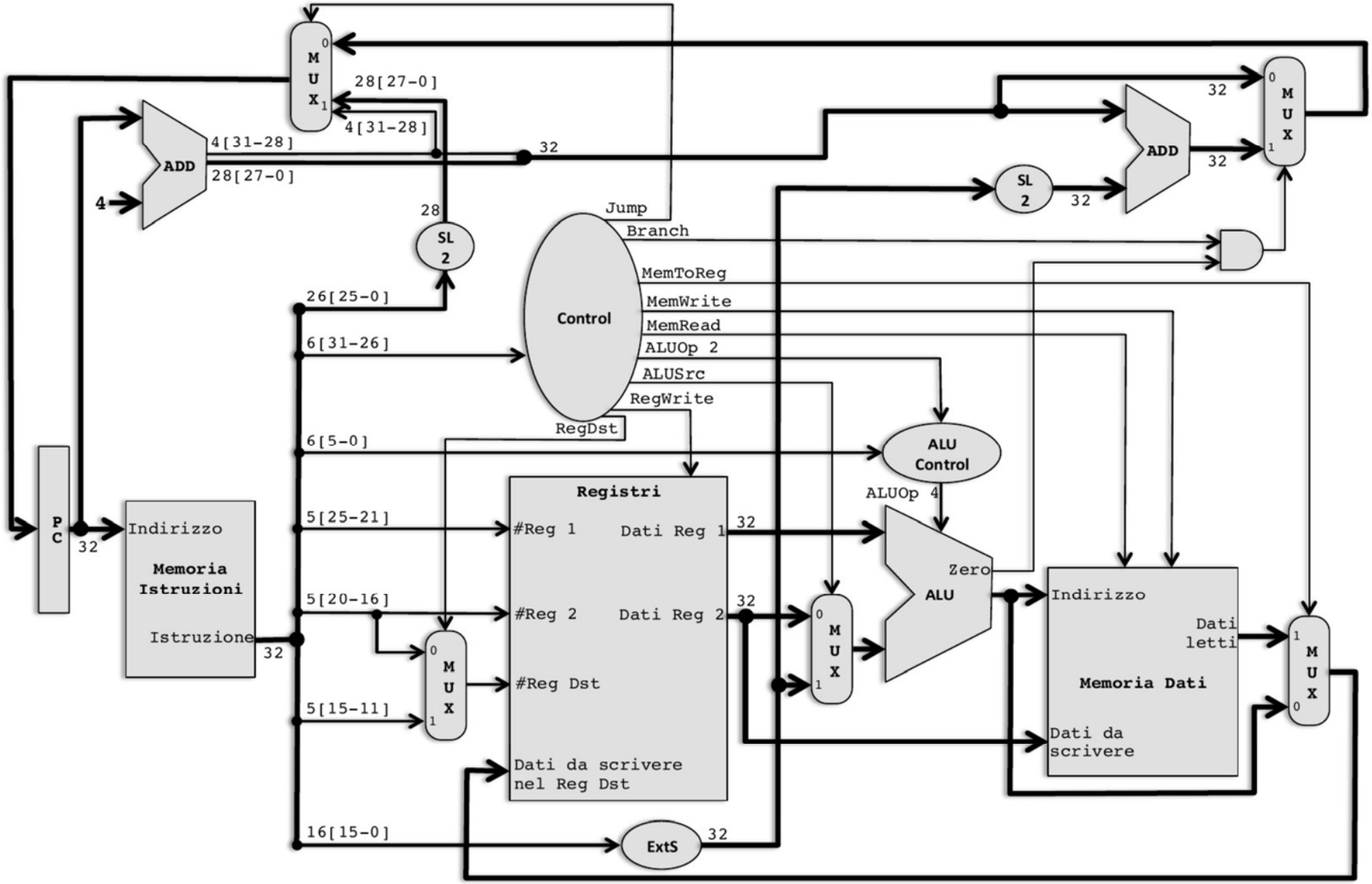
CPI = (n. stalli + n. istruzioni)/(n. istruzioni) = (21+11) / 11 = 2,9

Si proponga una nuova schedulazione (riordino) della sequenza di istruzioni contenuta nell'esercizio 3 in cui si ottenga il minor numero di conflitti possibile. Si consideri sempre una singola iterazione del ciclo eseguita dal processore MIPS in modalità pipeline a 5 stadi senza ottimizzazioni.
Individuare i conflitti sui dati di tipo RAW (Read After Write) e i conflitti sul controllo presenti nel nuova sequenza proposta.

n. stalli	TIPO CONFL.	ISTRUZIONE	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C15
		L1: beq $t6,$t7, EXIT	IF	ID	EX	ME	WB											
3	C	lw $t2,BASEA($t6)		IF	ID	EX	ME	WB										
		lw $t3,BASEB($t6)			IF	ID	EX	ME	WB									
2	D	add $t2,$t2,$t2				IF	ID	EX	ME	WB								
		add $t3,$t3,$t3					IF	ID	EX	ME	WB							
3	D	add $t4,$t2,$t3						IF	ID	EX	ME	WB						
		sw $t2,BASEA($t6)							IF	ID	EX	ME	WB					
		sw $t3,BASEB($t6)								IF	ID	EX	ME	WB				
1	D	sw $t4,BASEC($t6)									IF	ID	EX	ME	WB			
		addi $t6,$t6,4										IF	ID	EX	ME	WB		
		J L1											IF	ID	EX	ME	WB	
3	C	EXIT:												IF	ID	EX	ME	WB

CPI = (n. stalli + n. istruzioni)/(n. istruzioni) = (12+11) / 11 = 2,1

Considerate l'architettura MIPS a ciclo singolo dello schema sottostante.
Vogliamo aggiungere l'istruzione **`indj offset(rs)`** (indirect jump) che salta incondizionatamente all'indirizzo contenuto nella memoria nella posizione offset(rs), ovvero corrisponde a svolgere le due istruzioni `lw $at offset(rs) e jr $at.`
1) modificate il diagramma mostrando gli eventuali altri componenti necessari a realizzare l'istruzione
2) indicate sul diagramma i segnali di controllo necessari a realizzare l'istruzione
3) dire se è possibile il problema del conflitto sul controllo come nel caso dell'istruzione jump.



Si scriva in linguaggio C e poi tradurre in assembler del processore MIPS una procedura ricorsiva che calcola la funzione FIB(n) con n >=0 secondo la seguente definizione:

FIB(0) = 0
FIB (1) = 1
FIB (n) = FIB (n-1) * FIB (n-2) se n >=2

```
int FIB (int n) {
if ( n == 0 )
    return 0;
else if ( n < 2)
    return 1;
else
    return (FIB(n-1) + FIB(n-2));
}
```

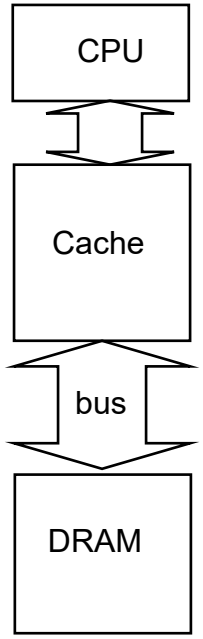
```
FIB:  addi $sp, $sp, -12    # alloca lo stack
      sw $ra, 8($sp)        # salvo return address
      sw $a0, 4($sp)        # salvo n
      sw $s1, 0($sp)        # salvo risultato parziale
IF:   bgtz $a0, ELSEIF      # se n > 0 salta a ELSEIF
      add $v0, $zero, $zero # se n==0 ritorna 0
      j END
ELSEIF: slti $t0, $a0, 2    #set $t0 <-1 se $a0 <2
      beq $t0, $zero, ELSE  # se $t0 == 0 salta a ELSE
      addi $v0, $zero, 1    # se n==1 ritorna 1
      j END
ELSE: addi $a0, $a0, -1     # $a0 <- n-1
      jal FUNC              # chiama f(n-1)
      move $s1, $v0         # salva f(n-1) in $t1
      addi $a0, $a0, -1     # $a0 <- n-2
      jal FUNC              # chiama f(n-2)
      add $v0, $v0, $s1     # f(n) <- f(n-1) * f(n-2)
END:  lw $ra, 8($sp)        # ripristino return address
      lw $a0, 4($sp)        # ripristino n
      lw $s1 0($sp)         # ripristino n
      addi $sp, $sp, 12     # dealloca lo stack
      jr $ra
```

Tradurre in linguaggio C la seguente porzione di codice assembler del processore MIPS.
I simboli VETTA, VETTB, VETTC, K1, e K2 sono costanti a 16 bit, prefissate.

add $t6, $zero, $zero	i ← 0;	i =0;
addi $t7, $zero, 400	N ← 100;	N =100;
L1: beq $t6, $t7, END	if (i == N) goto END;	while (i != N) {
lw $t2, VETTA($t6)	var2 ← VETTA[i];	
lw $t3, VETTB($t6)	var3 ← VETTB[i];	
L2: slt $t0, $t2, $t3	$t0 ←1 se var2 < var3 ;	
bne $t0, $0, L3	if $t0 != 0 goto L3;	
addi $t2, $t2, K1	var2 ← var2 + K1;	
sw $t2, VETTA($t6)	VETTA[i] ← var2;	
addi $t3, $t3, K2	var3 ← var3 + K2;	
sw $t3, VETTB($t6)	VETTB[i] ← var3;	
add $t4, $t2, $t3	var4 ← var2 + var3;	
sw $t4, VETTC($t6)	VETTC[i] ← var4;	
sw $zero, VETTD($t6)	VETTD[i] ← 0;	
j INC	goto INC;	else
L3: sw $zero, VETTC($t6)	VETTC[i] ← 0;	VETTC[i] = 0;
sub $t1, $t2, $t3	var1 ← var2 - var3;	
sw $t1, VETTD($t6)	VETTD[i] ← var1;	VETTD[i] = VETTA[i] - VETTB[i]; }
INC: addi $t6, $t6, 4	i++;	i++;
j L1	goto L1;	}
END:		

```
i =0;
N =100;
while (i != N) {
        if (vettA[i] >= vettB[i]) {
                VETTA[i] = VETTA[i] + K1;
                VETTB[i] = VETTB[i] + K2;
                VETTC[i] = VETTA[i] + VETTB[i];
                VETTD[i] = 0;
                }
        else {
                VETTC[i] = 0;
                VETTD[i] = VETTA[i] - VETTB[i];
                }
i++;
```

Si consideri un calcolatore con cache organizzata a blocchi di 4 parole ciascuno e si supponga che:



a. Il trasferimento dell'indirizzo vero la DRAM richieda un ciclo di clock
b. Il tempo necessario per accedere alla prima parola su una riga della DRAM richieda 12 cicli di clock
c. Il tempo necessario per accedere alle parole successive alla prima su una riga della DRAM richieda 6 cicli di clock
d. Il trasferimento di una parola dalla DRAM richieda- un ciclo di clock

Calcolare tempo necessario per trasferire le 4 parole di un blocco della cache nei seguenti 3 casi in cui le 4 parole si trovano:

1. su 4 righe diverse della DRAM (senza sovrapporre trasmissione di indirizzi e dati con lettura)

1 * 4	bus cycle per trasferimento indirizzo
12 *4	bus cycle per lettura
1 * 4	bus cycle per trasferimento dato
56	TOTALE

2. su 4 righe diverse della DRAM (sovrapponendo trasmissione di indirizzi e dati con lettura)

1	bus cycle per trasferimento indirizzo
12 *4	bus cycle per lettura
1	bus cycle per trasferimento dato
50	TOTALE

3. sulla stessa riga della DRAM

1	bus cycle per trasferimento indirizzo
12 + 6 * 3	bus cycle per lettura
1	bus cycle per trasferimento dato
32	TOTALE

4. in 4 banchi interallacciati di DRAM

1	bus cycle per trasferimento indirizzo
12	bus cycle per lettura
1 * 4	bus cycle per trasferimento dato
17	TOTALE

Un processore a 32 bit è dotato di una cache set associativa a due vie che utilizza i 32 bit di indirizzo nel modo seguente: 31-14 tag, 13-5 indice, 4-0 offset.

Calcolare:

1) La dimensione della linea di cache in numero di parole
   Dimensione linea di cache = 2^3 = 8 parole (restanti 3 bit offset tolti 2 bit per indirizzo byte)

2) La dimensione della cache complessiva in bit
   Dimensione cache = $2 * 2^9 * (1 + 18 + 8 * 2^5)$ = 281600 bit

In presenza della seguente sequenza di accessi in memoria (indirizzi riferiti al byte ed espressi in decimale):

indirizzo byte	17920	17948	149024	83488	17952	148998	149012	17964	148648	17848
modulo dimensione (indirizzo byte modulo 16384#)	1536	1564	1568	1568	1568	1542	1556	1580	1192	1464
Linea (diviso 32)	48	48	49	49	49	48	48	49	37	45
Tag (indirizzo byte diviso 16384)	1	1	9	5	1	9	9	1	9	1
	M	H	M	M	R	M	H	H	M	M

Dimensione parte dati cache in byte = $2^9 * 2^5$ = 16384 byte

Considerando la cache inizialmente vuota indicare per ciascun accesso se si tratta di hit, miss o replace e calcolare la frequenza di hit.
Frequenza HIT = 3/10 = 30%