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

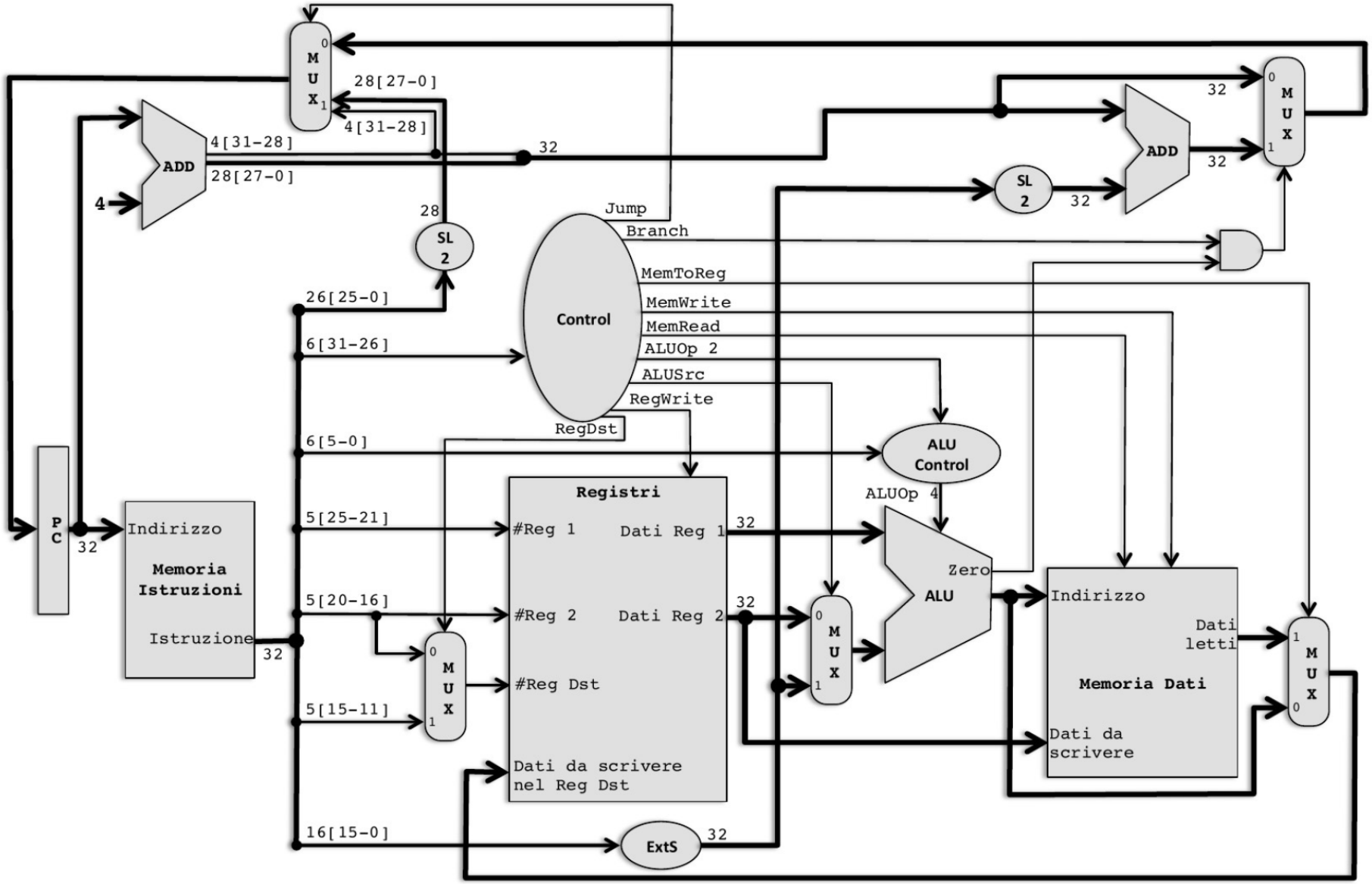
CPI = (n. stalli + n. istruzioni)/(n. istruzioni) = (21+11) / 11 = 2,9

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 f(n) con n >=0 secondo la seguente definizione:

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

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

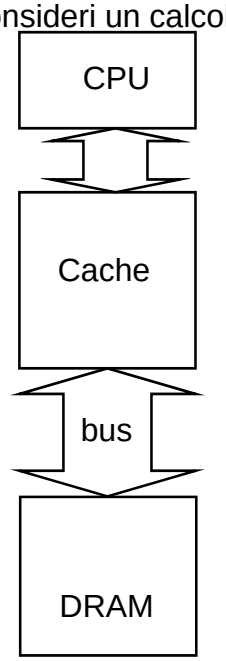
```
FUNC:   addi $sp, $sp, -12      # alloca lo stack
        sw $ra, 8($sp)          # salvo return address
        sw $a0, 4($sp)          # salvo n
        sw $s1, 0($sp)          # salvo risultato parziale
IF:     bgtz $a0, ELSEIF        # se n > 0 salta a ELSEIF
        add $v0, $zero, $zero   # 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)
        mul $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;
addi $t7, $zero, 400	N ← 100;
L1: beq $t6, $t7, END	if (i == N) goto END;
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;
L3: sw $zero, VETTC($t6)	VETTC[i] ← 0;
sub $t1, $t2, $t3	var1 ← var2 - var3;
sw $t1, VETTD($t6)	VETTD[i] ← var1;
INC: addi $t6, $t6, 4	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

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

2. 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

3. 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

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%

Corso di laurea in Ingegneria Informatica
Corso di laurea in Ingegneria delle Telecomunicazioni

Corso di Calcolatori Elettronici

Prova scritta del 9 febbraio 2018 - ARCHITETTURE

Cognome:	**Nome :**	**Matricola:**

ESERCIZIO 1 (10 punti)

Si scriva in linguaggio macchina del processore MIPS la seguente funzione di ordinamento:

```
void SelectionSort (int V[], int N) {
int i, j, min;

for (i=0; i<(N-1); i++) {
     min = i;
     for (j=i+1; j<N; j++)
          if (V[j] < V[min]) min = j;
     swap(V, i, min);
     }
}
```

ESERCIZIO 2 (8 punti)

Si consideri una gerarchia di memoria composta da memoria centrale da 4 GB indirizzabile per byte con parole da 32 bit, una memoria cache istruzioni a indirizzamento diretto (direct mapped) da 512 KB (per la sola parte dati) e una memoria cache dati set associativa a 2 vie da 1 MB (sempre per la sola parte dati) ed entrambe con blocchi (ovvero linee) da 256 byte. Il tempo di accesso alle parole della cache (dato e istruzione) è pari a 1 ciclo di clock. Il tempo di accesso alle parole di memoria centrale è pari a 10 cicli di clock.
Si chiede di:

1. Indicare la struttura degli indirizzi di memoria per le due memorie cache.
2. Calcolare la dimensione delle due memorie cache incluse le parti tag e bit di validità.
3. Calcolare il tempo necessario per caricare un blocco in caso di fallimento (miss).
4. Calcolare il tempo medio di accesso alla memoria per un programma dove in media il 25 % delle istruzioni eseguite richiede un accesso in lettura o scrittura a un dato. Il miss rate (frequenza di fallimento) della cache istruzioni è pari a 1% mentre per la cache dati è del 10%.

ESERCIZIO 3 (7 punti)

Sia dato il seguente ciclo di un programma in linguaggio macchina MIPS. Si supponga che i registri $6, e $7 siano stati inizializzati rispettivamente ai valori 0 e 4N. I simboli BASEA e BASEB sono costanti a 16 bit, prefissate. Il ciclo di clock del processore vale 5 ns.
Si consideri una generica iterazione del ciclo eseguita dal processore MIPS in modalità pipeline a 5 stadi senza ottimizzazioni.
Individuare i conflitti sui dati e i conflitti sul controllo presenti nel programma e indicarli nell'ultima colonna.
Inserire nella prima colonna il numero di stalli da inserire in modo da risolvere i conflitti presenti nel programma.

Num. stalli	ISTRUZIONE	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	Tipo conflitto
	L1:lw $2,BASEA($6)	IF	ID	EX	ME	WB												
	add $2,$2,$2		IF	ID	EX	ME	WB											
	add $2,$2,$2			IF	ID	EX	ME	WB										
	lw $3,BASEB($6)				IF	ID	EX	ME	WB									
	add $3,$3,$3					IF	ID	EX	ME	WB								
	add $3,$3,$3						IF	ID	EX	ME	WB							
	addi $6,$6,4							IF	ID	EX	ME	WB						
	sw $2,BASEA($6)								IF	ID	EX	ME	WB					
	sw $3,BASEB($6)									IF	ID	EX	ME	WB				
	bne $6,$7, L1										IF	ID	EX	ME	WB			
	EXIT:											IF	ID	EX	ME	WB		

Scrivere il numero totale di stalli inseriti nel programma e il CPI asintotico:

Si consideri inoltre una generica iterazione del ciclo dell'esercizio precedente eseguita dal processore MIPS nel quale siano state introdotte le seguenti ottimizzazioni:
- nel Register File è possibile la lettura e la scrittura allo stesso indirizzo nello stesso ciclo di clock;
- introduzione del forwarding
- calcolo del PC nelle branch sia stato anticipato nello stadio ID.

Scrivere il numero totale di stalli inseriti nel programma e il CPI asintotico:

ESERCIZIO 4 (8 punti)

Tradurre il seguente codice in linguaggio assembler nel corrispondente programma in linguaggio C:

```
func:
    addi $sp, $sp, -4
    sw   $s0, 0($sp)
    add  $s0, $zero,
L1: add  $t1, $s0, $a1
    lbu  $t2, 0($t1)
    add  $t3, $s0, $a0
    sb   $t2, 0($t3)
    beq  $t2, $zero, L2
    addi $s0, $s0, 1
    j    L1
L2: lw   $s0, 0($sp)
    addi $sp, $sp, 4
    jr   $ra
```

Corso di laurea in Ingegneria Informatica
Corso di laurea in Ingegneria delle Telecomunicazioni

Corso di Calcolatori Elettronici

Prova scritta del 15 gennaio 2018 - ARCHITETTURE

Cognome:	**Nome :**	**Matricola:**

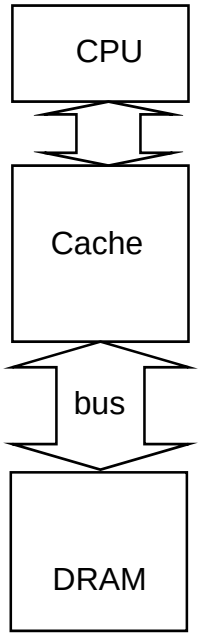
ESERCIZIO 1 (10 punti)

Si scriva in assembler del processore MIPS la seguente funzione per l'ordinamento di un vettore di numeri interi:

```
void InsertionSort (ItemType a[], int N) {
int i,j;
ItemType v;

for (i=1; i<N; i++) {
      v = a[i];
      j = i;
      while (a[j-1] > v && j>0) {
            a[j] = a[j-1];
            j--;
            }
      a[j] = v;
      }
}
```

ESERCIZIO 2 (7 punti)



Si consideri un calcolatore con cache organizzata a blocchi di 4 parole ciascuno.

Si supponga inoltre 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
2. sulla stessa riga della DRAM
3. in 4 banchi interallacciati di DRAM

ESERCIZIO 3 (7 punti)

Considerate l'architettura MIPS con pipeline e la sequenza di istruzioni che copia in un vettore gli elementi dispari di un vettore riportata in tabella.
Considerate che il vettore contenga 100 elementi di cui 30 con valore pari e 70 con valore dispari (l'ordine non è rilevante).
Indicate:

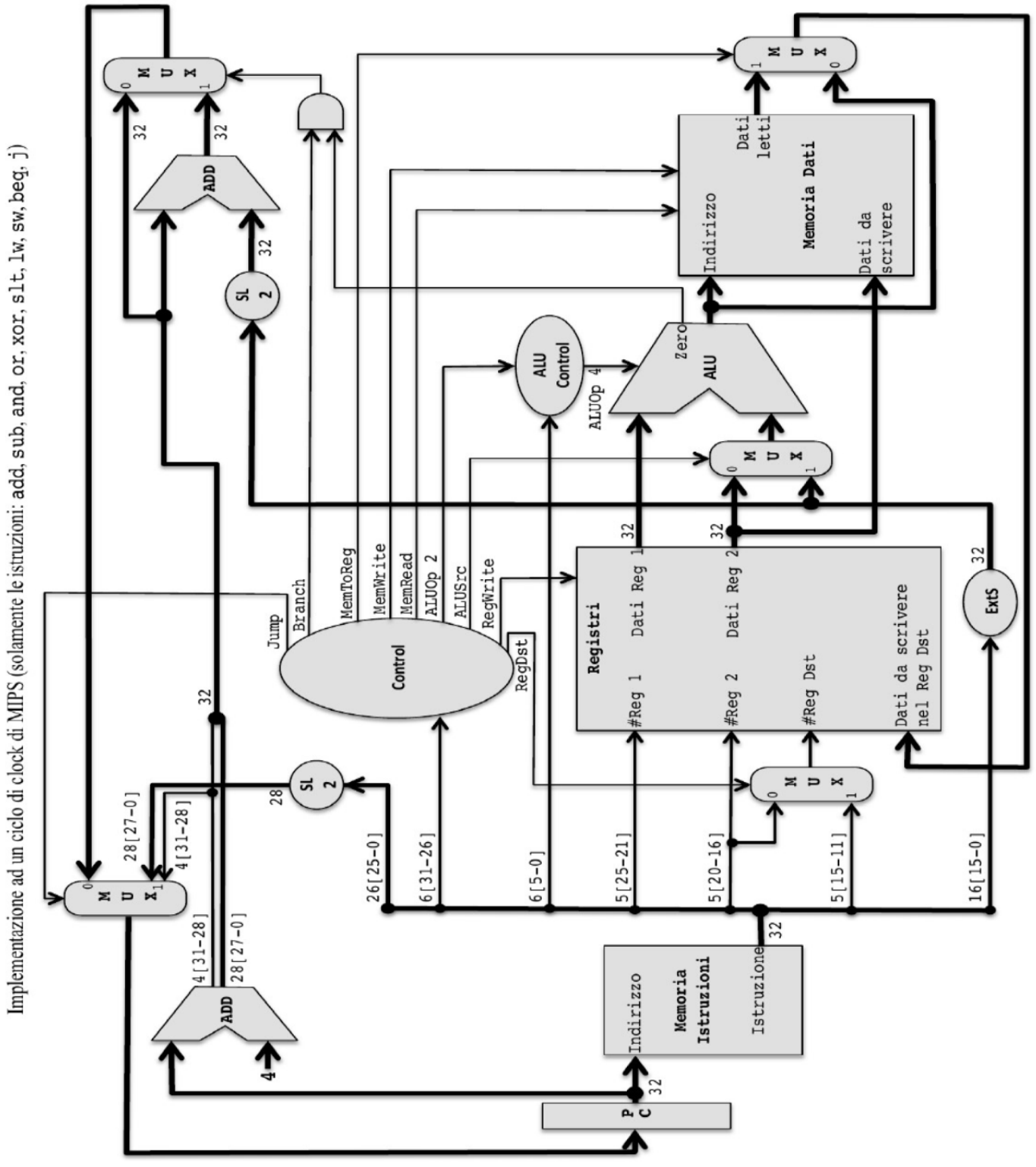
1) tra quali istruzioni sono presenti conflitti su dati e controlli e quanti stalli sono necessari nei due casi:
   A senza ottimizzazioni
   B con ottimizzazioni sia per i conflitti sui dati che sui controlli
2) quanti cicli di clock sono necessari a eseguire tutto il programma
   A senza ottimizzazioni
   B con ottimizzazioni sia per i conflitti sui dati che sui controlli

Stalli A	Stalli B	ISTRUZIONE	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	Tipo conflitto
		`move $s0, $zero`	IF	ID	EX	ME	WB												
		`li $s1, -4`		IF	ID	EX	ME	WB											
		`lw $s2, DIM`			IF	ID	EX	ME	WB										
		`sll $s2, $s2, 2`				IF	ID	EX	ME	WB									
		`loop: beq $s0, $s2, fine`					IF	ID	EX	ME	WB								
		`lw $t2, vettore1($s0)`						IF	ID	EX	ME	WB							
		`andi $t3, $t2, 1`							IF	ID	EX	ME	WB						
		`beqz $t3, next`								IF	ID	EX	ME	WB					
		`addi $s1, $s1, 4`									IF	ID	EX	ME	WB				
		`sw $t2, vettore2($s1)`										IF	ID	EX	ME	WB			
		`next: addi $s0,$s0,4`											IF	ID	EX	ME	WB		
		`j loop`												IF	ID	EX	ME	WB	
		`fine:`													IF	ID	EX	ME	

ESERCIZIO 4 (7 punti)

Data l'architettura MIPS **a ciclo singolo** in figura, si vuole aggiungere l'istruzione di tipo I **bali rd, label** (branch and link immediato) che salta all'indirizzo della label (relativo come nei beq) e salva nel registro **rd** l'indirizzo della prossima istruzione.

1) modificare il diagramma inserendo eventuali altri componenti necessari a realizzare l'istruzione.
2) indicare sul diagramma i segnali di controllo necessari a realizzare l'istruzione
3) supponendo che l'accesso alle memorie impieghi 133ns, l'accesso ai registri 66ns, le operazioni dell'ALU 200ns, e ignorando gli altri ritardi di propagazione dei segnali, indicare sul diagramma la durata totale del ciclo di clock per permettere l'esecuzione anche della nuova istruzione.



Corso di laurea in Ingegneria Informatica
Corso di laurea in Ingegneria delle Telecomunicazioni

Corso di Calcolatori Elettronici

Prova scritta del 21 dicembre 2017 - ARCHITETTURE

Cognome:	**Nome :**	**Matricola:**

ESERCIZIO 1—

Si scriva in assembler del processore MIPS la funzione fattoriale nelle due versioni ricorsiva e iterativa:

```
int fattoriale(int numero){
  int f;
  if (!numero)
    f=1;
  else
    f=numero*fattoriale(numero-1);
  return f;
}

int fattoriale(int numero){
  int i,f;
  f=1;
  for(i=numero;i>0;i--)
    f *= i;
  return f;
}
```

ESERCIZIO 2

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
2) La dimensione della cache complessiva in bit

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

17920 17948 149024 83488 17952 148998 149012 17964 148648 17848

Considerando la cache inizialmente vuota indicare per ciascun accesso se si tratta di hit, miss o replace e calcolare la frequenza di hit.

ESERCIZIO 3

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 VA, VB, e VC sono costanti a 16 bit, prefissate. Il ciclo di clock del processore vale 2 ns.
Si consideri una singola iterazione del ciclo eseguita dal processore MIPS in modalità pipeline a 5 stadi con forwarding.
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	ISTRUZIONE	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C15	Tipo conflitto
	L1: beq $t6,$t7, END	IF	ID	EX	ME	WB												
	lw $t2,VA($t6)		IF	ID	EX	ME	WB											
	add $t2,$t2,$t2			IF	ID	EX	ME	WB										
	sw $t2,VA($t6)				IF	ID	EX	ME	WB									
	lw $t3,VB($t6)					IF	ID	EX	ME	WB								
	add $S3,$t3,$t3						IF	ID	EX	ME	WB							
	add $S4,$t2,$t3							IF	ID	EX	ME	WB						
	sw $S3,VB($t6)								IF	ID	EX	ME	WB					
	sw $S4,VC($t6)									IF	ID	EX	ME	WB				
	addi $t6,$t6,4										IF	ID	EX	ME	WB			
	J L1											IF	ID	EX	ME	WB		
	END:												IF	ID	EX	ME	WB	

Scrivere il numero totale di stalli inseriti nel programma:

Calcolare il CPI (numero di cicli di clock per istruzione) asintotico ottenuto:

ESERCIZIO 4

Si ha il dubbio che in una partita di CPU a ciclo di clock singolo la Control Unit sia rotta, producendo il segnale di controllo **RegWrite** attivo **anche quando** è attivo il segnale di controllo **MemWrite**.

Si indichino quali delle istruzioni **lw, sw, add, sub, and, or, xor, slt, beq, j** funzioneranno male e perché fornendo un esempio di malfunzionamento.