Esercizio 1 (11 punti).

L1 #BL = ADR/8 IND = #BL % 4 TAG = #BL/4

Sia data una CPU con processore a **2GHz** e **4 CPI** (Clock per Instruction) che adoperi indirizzi da 32 bit e memoria strutturata su due livelli di **cache** (L1, L2), il cui setup è come segue:
L1 è una cache **set-associativa** a **3 vie** con **4 set** e **blocchi da 2 word**; adopera una politica di rimpiazzo **LRU**. Ricordiamo che consideriamo il set come l'insieme del blocco in cache con tag e bit di validità, mentre la linea è il gruppo di set con il medesimo indice.
L2 è una cache **direct-mapped** con **4 linee** e **blocchi da 16 word**.

L2 = #BL: ADR/64 IND #BL % 4 TAG: #BL/4

1) Supponendo che all'inizio nessuno dei dati sia in cache, indicare quali degli accessi in memoria indicati di seguito sono HIT o MISS in ciascuna delle due cache. Per ciascuna **MISS** indicare se sia di tipo Cold Start (**Cold**), Capacità (**Cap**) o Conflitto (**Conf**). Utilizzare la tabella sottostante per fornire i risultati ed indicare la metodologia di calcolo più in basso.

	Address	128	8	162	40	132	12	192	48	4096	160	2096	480
L1	Block#	16	1	20	5	16	1	24	6	512	20	262	60
	Index	0	1	0	1	0	1	0	2	0	0	2	0
	Tag	4	0	5	1	4	0	6	1	128	5	65	15
	HIT/MISS	MISS	MISS	MISS	MISS	HIT	HIT	MISS	MISS	MISS	MISS	MISS	MISS
	Miss type	COLD	COLD	COLD	COLD			COLD	COLD	COLD	CONFLICT	COLD	COLD
L2	Block#	2	0	2	0			3	0	64	2	32	7
	Index	2	0	2	0			3	0	0	2	0	3
	Tag	0	0	0	0			0	0	16	0	8	1
	HIT/MISS	MISS	MISS	HIT	HIT			MISS	HIT	MISS	HIT	MISS	MISS
	Miss type	COLD	COLD					COLD		COLD		COLD	COLD

2) Calcolare le dimensioni in bit (compresi i bit di controllo ed assumendo che ne basti uno per la LRU) delle due cache: (a) L1 e (b) L2.

3) Assumendo che gli accessi in **memoria** impieghino **400 ns**, che gli **hit** nella cache **L1** impieghino **1 ns** e gli **hit** nella cache **L2** impieghino **40 ns**, calcolare (a) il **tempo totale** per la sequenza di accessi, (b) il tempo **medio** per la sequenza di accessi, e (c) **quante istruzioni** vengono svolte nel tempo medio calcolato.

4) Calcolare il word offset del primo indirizzo (128) per la cache L1 spiegando i calcoli effettuati.

5) Considerando i tipi di miss che possono verificarsi durante l'uso della memoria virtuale, indicare se possa presentarsi il caso in cui si abbiano sia TLB hit sia page table miss, spiegando brevemente il motivo. Si noti che questa è una domanda teorica non collegata alla domanda (1) di questo esercizio.

(2) $L1 = 4 \cdot 3 \cdot (64 + 1 + (32 - 5)) = 1104$ bit

$L2 = 4 \cdot (512 + 1 + (32 - 4 - 4)) = 2148$ bit

(3) $TOT = 6 \cdot 400 + 2 + 40 \cdot 4 = 2562$ ns

TEMPO MEDIO $\Rightarrow \frac{2562}{12} = 213.5$ ns

COLPI DI CLOCK = 427 cc

ISTRUZIONI MEDIE $= \frac{427}{4} = 106.75$

(4) WORD OFFSET DI 128 = 0

Esercizio 2 (11 punti).

Considerare l'architettura MIPS a ciclo singolo nella figura in basso (ed in allegato).
Si vuole aggiungere l'istruzione di **tipo I** "branch if less than (safe)"
`bltsafe $rs, $rt, offset`
che richiede il salto all'istruzione il cui indirizzo è in `PC + 4 + offset` se e solo se valgono ambedue le seguenti condizioni:
a) `$rs < $rt`;
b) `PC + 4 + offset ≤ 0x7FFFFFFF`.

Si specifica che `offset` è la parte immediata dell'istruzione e viene usato come indirizzo *relativo* di destinazione.

1) Mostrare le **modifiche all'architettura** della CPU MIPS, avendo cura di aggiungere eventuali altri componenti necessari a realizzare l'istruzione. A tal fine, si può alterare la stampa del diagramma architetturale oppure ridisegnare i componenti interessati dalla modifica, avendo cura di indicare i fili di collegamento e tutti i segnali entranti ed uscenti. Indicare inoltre sul diagramma i **segnali di controllo** che la CU genera *per realizzare l'istruzione*.

2) Indicare il contenuto in bit della word che esprime l'istruzione
`bltsafe $16, $19, 0x0FFA`
compilando la tabella sottostante (assumiamo che lo *OpCode* di `bltsafe` sia `0x38`).

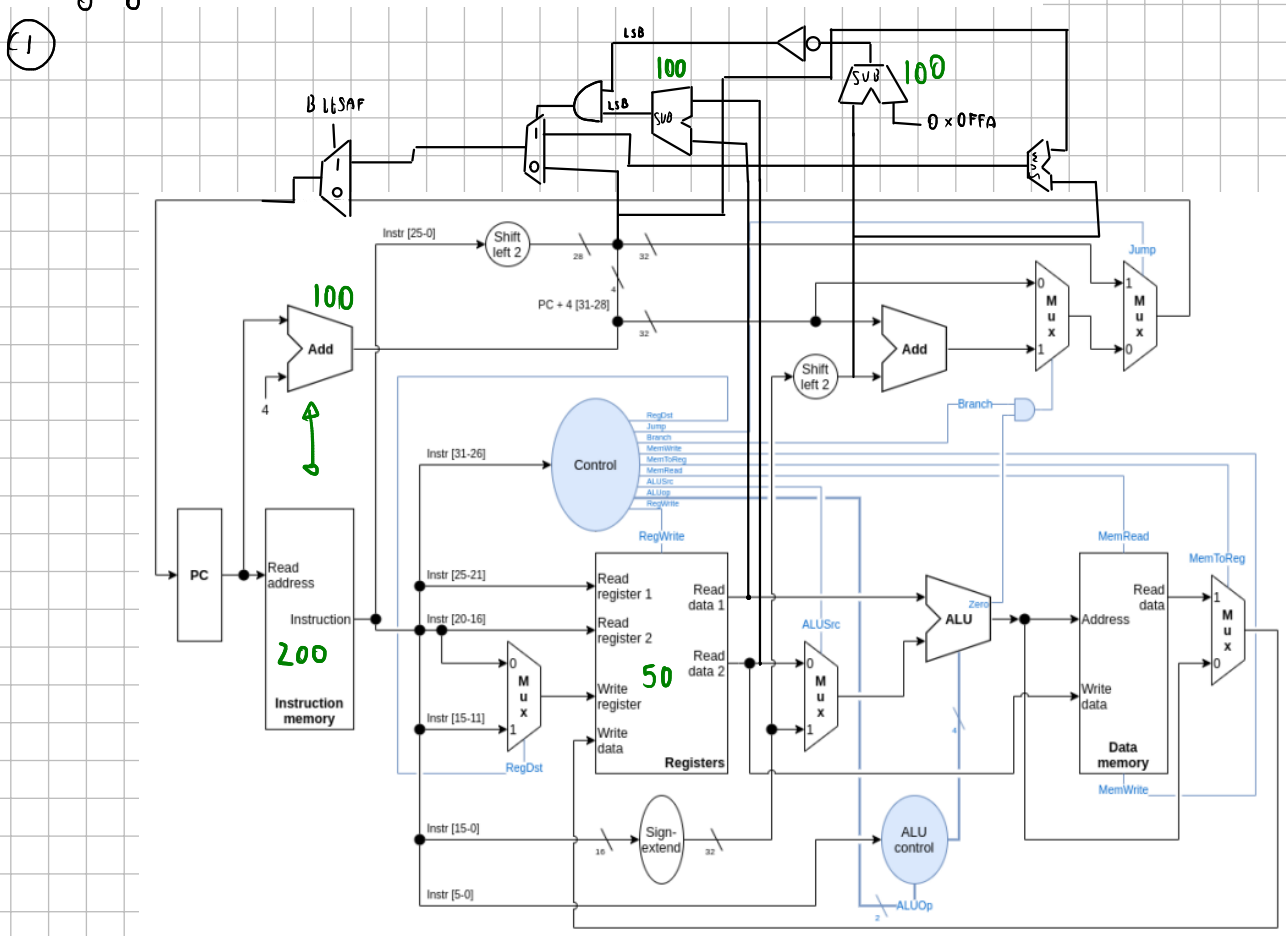
1	1	1	0	0	0	1	0	0	0	0	1	0	0	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	0	1	0

3) Supponendo che l'accesso alle **memorie** impieghi **200 ns**, l'accesso ai **registri 50 ns**, le operazioni dell'**ALU e** dei **sommatori 100 ns**, e che gli altri ritardi di propagazione dei segnali siano trascurabili, indicare la durata totale del **ciclo di clock** tale che anche l'esecuzione della nuova istruzione sia permessa *spiegando i calcoli effettuati*.

4) Indicando con `Blts` il segnale di controllo che viene asserito per eseguire la nuova istruzione, assumiamo che
a) tutti i segnali di tipo *don't care* siano pari a `0` e che
b) la Control Unit della CPU MIPS modificata per supportare `bltsafe` sia difettosa e sovrascriva il segnale `RegWrite` come segue:
`RegWrite = Blts`
In tal caso, quale sarà il valore di `$s1` al termine dell'esecuzione del seguente frammento di codice assumendo che `$s0` e `$s1` siano inizializzati a `0`? Spiegare il motivo.

```
addi    $s0, $0, 2
addi    $s1, $0, 1
bltsafe $s0, $s1, 0x0FFA
```

0 0



REGDST	JUMP	BRANCH	MEMWRITE	MEMTOREG	MEMREAD	ALUSRC	ALUOP	REGWRITE	BLTSAFE
X	X	X	0	X	X	X	XX	0	1

(3) T = 200 + 50|100 = 350 ns
50 | 100

(4)
REGWRITE = BLST $S1 = 0

il risultato di S1 è 0, perché l'add NON FUNZIONA, quindi
blts = 0 QUINDI REGWRITE = 0

Esercizio 3 (11 punti)

Si consideri l'architettura MIPS con pipeline mostrata nella figura in basso (ed in allegato). Il programma qui di seguito (fornito sia in formato testo che come immagine) effettua la somma dei valori assoluti (in $s0) degli elementi presenti nell'array A di lunghezza L. Se il risultato è maggiore di 10, viene stampata la stringa S.

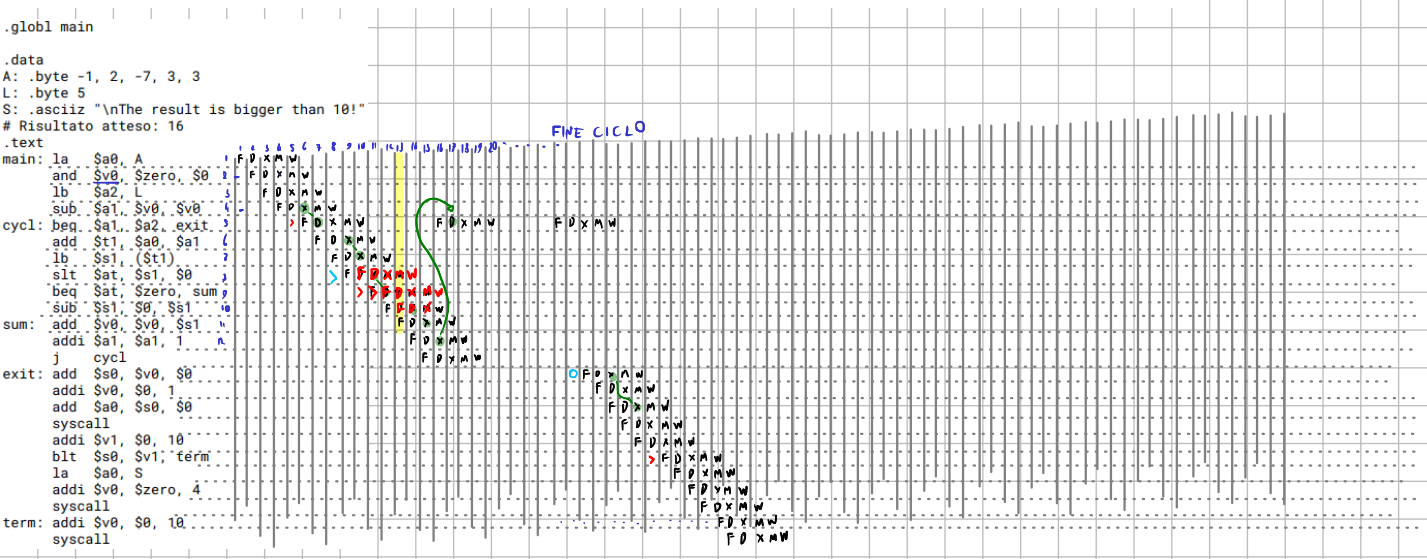
Si noti che, dato l'input fornito, il risultato atteso è 16 e per ciò la stringa viene stampata.

```
1  .globl main
2
3  .data
4  A: .byte -1, 2, -7, 3, 3
5  L: .byte 5
6  S: .asciiz "\nThe result is bigger than 10!"
7  # Risultato atteso: 16
8  .text
9  main: la   $a0, A # Indirizzo base array input
10       and  $v0, $zero, $0 # $v0: risultato
11       lb   $a2, L # Lunghezza dell'array
12       sub  $a1, $v0, $v0  # $a1: Indice corrente
13 cycl: beq  $a1, $a2, exit # Array scorso? Esci
14       add  $t1, $a0, $a1  # $t1: Indirizzo corrente
15       lb   $s1, ($t1) # $s1: Elemento corrente
16       slt  $at, $s1, $0 # $at vale 1 se $s1 < 0
17       beq  $at, $zero, sum # Se $s1 ≥ 0, somma
18       sub  $s1, $0, $s1 # Altrimenti, inverti segno
19 sum:  add  $v0, $v0, $s1 # Somma $s1 in v.assoluto
20       addi $a1, $a1, 1 # Passa a cella successiva
21       j    cycl # Riprendi il ciclo
22 exit: add  $s0, $v0, $0 # Salva risultato in $s0
23       addi $v0, $0, 1 # Imposta stampa interi
24       add  $a0, $s0, $0 # Imposta stampa ris.
25       syscall # Stampa
26       addi $v1, $0, 10 # $v1 = 10
27       blt  $s0, $v1, term # Se $s0 > 10, stampa
28       la   $a0, S # Carica la stringa S
29       addi $v0, $zero, 4 # Imposta stampa
30       syscall # Stampa
31 term: addi $v0, $0, 10 # Imposta uscita
32       syscall # Esci
```

per

```
.globl main

.data
A: .byte -1, 2, -7, 3, 3
L: .byte 5
S: .asciiz "\nThe result is bigger than 10!"
# Risultato atteso: 16
.text
main: la   $a0, A
      and  $v0, $zero, $0
      lb   $a2, L
      sub  $a1, $v0, $v0
cycl: beq  $a1, $a2, exit
      add  $t1, $a0, $a1
      lb   $s1, ($t1)
      slt  $at, $s1, $0
      beq  $at, $zero, sum
      sub  $s1, $0, $s1
sum:  add  $v0, $v0, $s1
      addi $a1, $a1, 1
      j    cycl
exit: add  $s0, $v0, $0
      addi $v0, $0, 1
      add  $a0, $s0, $0
      syscall
      addi $v1, $0, 10
      blt  $s0, $v1, term
      la   $a0, S
      addi $v0, $zero, 4
      syscall
term: addi $v0, $0, 10
      syscall
```



FINE CICLO + USCITA = 14

4 + 5 + 14 + 11 · 5 = 78