Esercizio 1A (11 punti).

Sia data una CPU con processore a **24GHz** e **16 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 **8 vie** con **2 set** e **blocchi da 16 word**; adopera una politica di rimpiazzo **LRU**. Ricordiamo che consideriamo la linea come l'insieme del blocco in cache con tag e bit di validità, mentre il set è il gruppo di linee con il medesimo indice.
L2 è una cache **direct-mapped** con **8 linee** e **blocchi da 128 word**.

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 e indicare la metodologia di calcolo.

	Address	2000	2124	8082	200	8512	8680	9048	264	216	264	320	2024
L1	Block#												
	Index												
	Tag												
	HIT/MISS												
	Miss type												
L2	Block#												
	Index												
	Tag												
	HIT/MISS												
	Miss type												

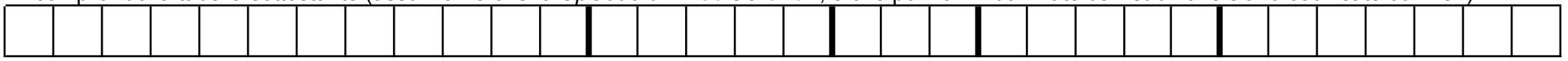
2) Calcolare le dimensioni in bit (compresi i bit di controllo ed assumendo che ne basti uno per la LRU) delle due cache: (a) L1 e (b) L2.
3) Assumendo che gli accessi in **memoria** impieghino **400 ns**, che gli **hit** nella cache **L1** impieghino **10 ns** e gli **hit** nella cache **L2** impieghino **20 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 sesto indirizzo (8680) per la cache L2 spiegando i calcoli effettuati.
5) Supponendo che gli indirizzi nella tabella siano virtuali e la memoria virtuale consti di **512 pagine** di **4KiB** ciascuna, indicarne i numeri di pagina virtuale.

Address	2000	2124	8082	200	8512	8680	9048	264	216	264	320	2024
Page#												

Esercizio 2 (11 punti).

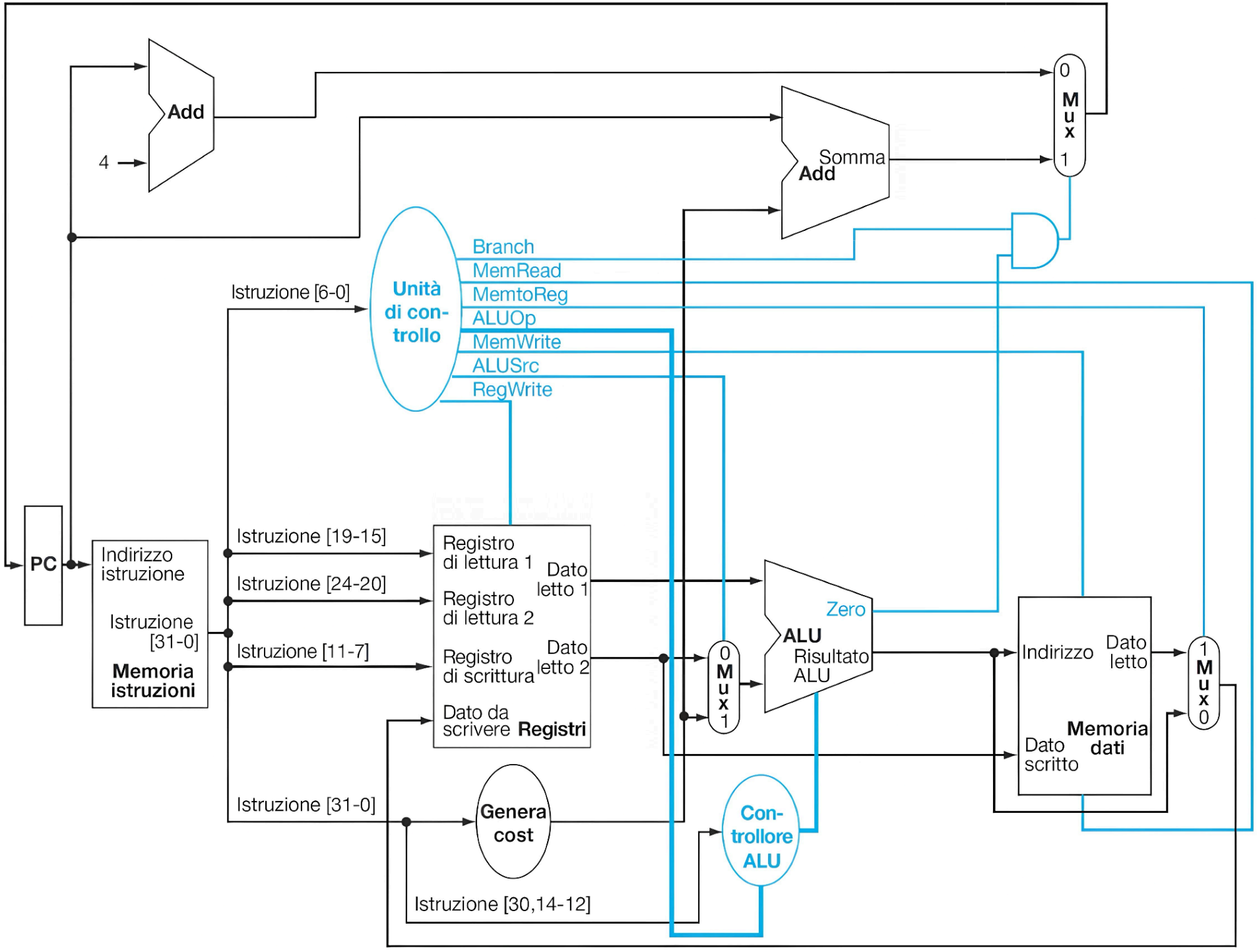
Considerare l'architettura RISC-V a ciclo singolo nella figura in basso (e in allegato).
Si vuole aggiungere alla CPU l'istruzione **load word if greater** (`lwig`), di **tipo I** e sintassi assembly:
`lwig` **`outreg, val1, val2_address`**
(dove `outreg`, `val1` e `val2_address` sono rispettivamente i campi `rd`, `rs1`, e il campo immediate dell'istruzione). L'istruzione deve operare come segue:
a) carica dal banco registri `val1`;
b) estenda il segno di **`val2_address`** ottenendo una word che chiameremo $\mathbf{val2\_address}_{32}$
c) carica da memoria la word all'indirizzo ($\mathbf{val2\_address}_{32}$ × 4) + **`0x1001 0000`**; nel seguito, chiameremo questa word **`val2`**;
d) compara **`val1`** (ottenuto dal banco registri) a **`val2`** (letto da memoria);
e) se **`val1 > val2`** salva nel registro di indice **`outreg`** il valore **`val1`**; altrimenti, salva nel registro di indice **`outreg`** il valore **`val2`**.
Esempio: Supponiamo che **`val1`** sia `s4`, contenente `0x00AA 0FD6`, che **`outreg`** sia `t4`,
che **`val2_address`** valga `0x0203` e che in memoria, all'indirizzo **`0x1001 080C`**, ci sia la word `0x00AA 00DA`.
In tal caso, abbiamo i seguenti risultati:
`val1` = `0x00AA 0FD6`
(`0x0203 × 4`) + `0x1001 0000` = `0x1001 080C`. Sappiamo (in questo esempio) che la word salvata in memoria a quell'indirizzo: è `0x00AA 00DA`.
Dunque, **`val2`** = `0x00AA 00DA`. Ne consegue che **`val1 > val2`** Dunque, viene salvato `val1` nel registro `t4`, ossia `t4 = 0x00AA 0FD6`.

1) Mostrare le **modifiche all'architettura** della CPU RISC-V, 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
   `lwig` **`t4, s4, 0x0203`**
   compilando la tabella sottostante (assumiamo che lo *OpCode* di `wvss` sia `0x3D`, e che porzioni inutilizzate dall'istruzione siano codificate con zeri).



3) Supponendo che l'accesso alle **memorie** impieghi **200 ns**, l'accesso ai **registri 50 ns**, le operazioni dell'**ALU e** dei **sommatori 150 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 `Lwig` 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 RISC-V modificata per supportare `Lwig` sia difettosa e sovrascriva il segnale `RegWrite` come segue:
   `RegWrite` ≔ `MemWrite` (il simbolo ≔ denota che la variabile a sinistra assume il valore dato della variabile a destra)
   In tal caso, indicare quale valore sia assegnato a `a7` al termine dell'esecuzione del seguente frammento di codice, assumendo che i registri `a7`, `s0` e `s1` siano inizializzati a `0`. Motivare la propria risposta.

```
      addi s0, s0, 0x8004
      beq  s1, s0, Out
      jal  x0, Exit
Out:  add  a7, zero, s1
Exit: addi a7, a7, 0x8002
```


Add
4
0
M
u
x
1
Somma
Add
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Unità di con-trollo
Istruzione [6-0]
PC
Indirizzo istruzione
Istruzione [31-0]
Memoria istruzioni
Istruzione [19-15]
Istruzione [24-20]
Istruzione [11-7]
Registro di lettura 1
Registro di lettura 2
Registro di scrittura
Dato da scrivere
Dato letto 1
Dato letto 2
Registri
Zero
ALU
Risultato ALU
Indirizzo
Dato letto
Dato scritto
Memoria dati
Istruzione [31-0]
Genera cost
Con-trollore ALU
Istruzione [30,14-12]

Esercizio 3 (11 punti)

Si consideri l'architettura RISC-V con pipeline. Il programma qui di seguito effettua la somma dei soli valori dispari (dopo averli moltiplicati per due) degli elementi presenti nell'array `vettore` di lunghezza 10. Infine, viene stampato il valore di tale somma.

```
1   .data
2   vettore: .word 24, 1, 46, 54, 50, 12, 2, 11, 39, 4
3    # 7 pari, 3 dispari
4
5   .text                       # Sommo solo i dispari moltipl. per 2
6   main:   addi t0, zero, 0    # somma parziale
7           la   s0, vettore    # indirizzo base
8           addi t1, s0, 36     # offset ultimo dei 40 byte
9   ciclo:  lw   t3, 0(t1)
10          andi t2, t3, 1      # è dispari?
11          beq  t2, zero, salta # se non lo è, salto
12          slli t3, t3, 1      # altrimenti moltiplico per 2 il numero
13          add  t0, t0, t3     # e lo accumulo
14  salta:  addi t1, t1, -4     # prossimo elemento
15          bge  t1, s0, ciclo  # fine del ciclo?
16
17          addi a7,x0, 1       # stampa…
18          mv a0, t0           # … la somma
19          ecall
```

Si supponga che non si faccia uso di alcuna pseudoistruzione, e che le ecall non richiedano alcuno stallo. Si consideri che la decisione di branch venga presa in fase di ID, e che le unità di forwarding (quando usate) includano il forwarding da EXE a EXE, da MEM a EXE, e (per il branch) da EXE a ID.

Si indichino (ignorando hazard che possano concernere la ecall):

1) Per i primi 15 cicli di clock (**senza unità di forwarding**), le istruzioni tra le quali sono presenti **data hazard** – indicare i numeri di linea N ed M (visibili alla sinistra delle linee di codice in figura) ed il registro coinvolto xY (nel formato N / M / xY, ad esempio 17 / 18 / t0 se c'è un data hazard causato dal registro t0 tra l'istruzione alla linea 17 e quella alla linea 18);

2) Per i primi 15 cicli di clock (**con unità di forwarding**) le istruzioni tra le quali sono presenti **control hazard** – indicare i numeri di linea N ed M (nel formato N / M, ad esempio se l'istruzione alla linea 17 causa un control hazard con l'istruzione alla linea 23 scrivere: 17 / 23);

3) quanti **cicli di clock** sono necessari ad eseguire il programma tramite **forwarding**, spiegando il calcolo effettuato;

4) quanti **cicli di clock** sarebbero necessari ad eseguire il programma **senza forwarding**, spiegando il calcolo effettuato;

5) quali sono, per ognuna delle cinque fasi, le **istruzioni** (o le bolle) in pipeline durante il **15° ciclo di clock** (**con forwarding**);
   IF:
   ID:
   EX:
   MEM:
   WB:

6) **Domanda teorica sulla pipeline qui**