Esame di Architettura degli Elaboratori, a.a. 2022/2023, appello straordinario 20/03/2023

Esercizio 1 (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 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 **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#	31	33	126	3	133	135	141	4	3	4	5	31
	Index	1	1	0	1	1	1	1	0	1	0	1	1
	Tag	15	16	63	1	66	67	70	2	1	2	2	15
	HIT/MISS	MISS	MISS	MISS	MISS	MISS	MISS	MISS	MISS	HIT	HIT	MISS	HIT
	Miss type	COLD	COLD	COLD	COLD	COLD	COLD	COLD	COLD			COLD	
L2	Block#	3	4	15	0	16	16	17	0			0	
	Index	3	4	7	0	0	0	1	0			0	
	Tag	0	0	1	0	2	2	2	0			0	
	HIT/MISS	MISS	MISS	MISS	MISS	MISS	HIT	MISS	MISS			HIT	
	Miss type	COLD	COLD	COLD	COLD	COLD		COLD	CONF				

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. Si assume che la cache sia a monte della memoria virtuale.

Address	2000	2124	8082	200	8512	8680	9048	264	216	264	320	2024
Page#	0	0	1	0	2	2	2	0	0	0	0	0

DIMENSIONI CACHE

L1
BLOCCO 512 +
V.bit 1 +
LRU 1 +
TAG 24 =
538 X
VIE·SET 16 = 8608 bit

L2
BLOCCO 4096 +
V.bit 1 +
~~LRU~~ +
TAG 20 =
4117 X
LINEE 8 = 32936 bit

TEMPI

TEMPO TOTALE = $400 \cdot 7 + 10 \cdot 3 + 20 \cdot 2 = 2870$ ns

TEMPO MEDIO = $2870 / 12 = 239.166$ ns

COLPI DI CLOCK MEDI = $239.166 \cdot 24 = 5739.984$ C.C.

ISTRUZIONI MEDIE = $5739.984 / 16 = 358,749$ ISTRUZIONI

WORD OFFSET

INDIRIZZO = 8680

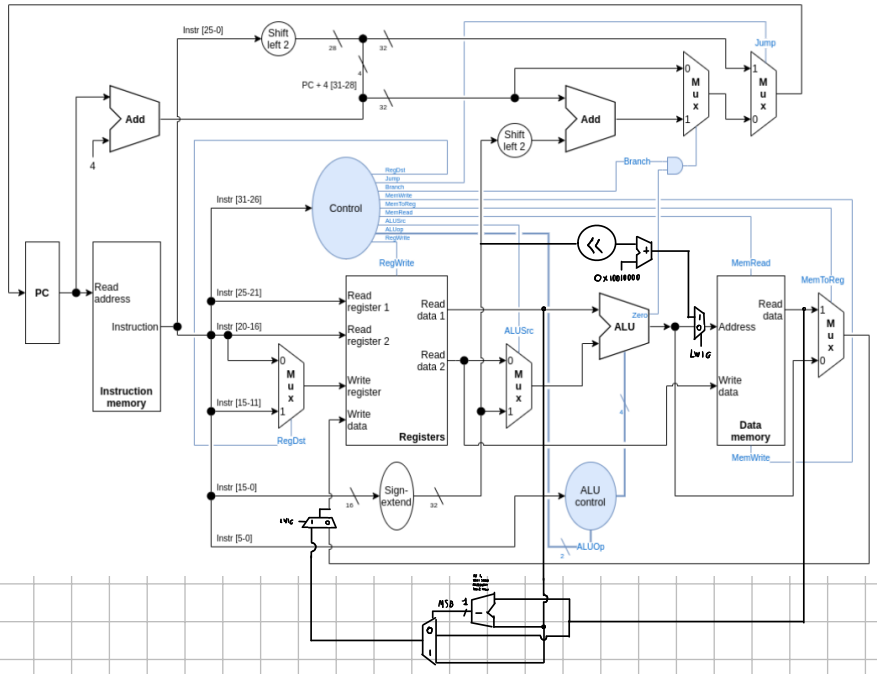
BYTE PER BLOCCO (L2) = 512

$$\left\lfloor \frac{\text{INDIRIZ. \% BYTE PER BLOCCO}}{4} \right\rfloor = \left\lfloor \frac{8680 \,\%\, 512}{4} \right\rfloor = 122$$

Esercizio 2 (11 punti).

Considerare l'architettura MIPS 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 $val1, $outreg, val2_address`
(dove `val1`, `outreg` e `val2_address` sono rispettivamente i campi `rs`, `rt` e il campo immediate dell'istruzione). L'istruzione deve operare come segue:
a) carica dal banco registri `$val1`;
b) carica da memoria la word all'indirizzo (**`val2_address` × 4) + `0x1001 0000`**; nel seguito, chiameremo questa word **`$val2`**;
c) compara **`$val1`** (ottenuto dal banco registri) a **`$val2`** (letto da memoria);
d) se **`$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`. Conosciamo per ipotesi in alto 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 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*.



REGDST = 0
JUMP = 0
BRANCH = 0
MEMWRITE = 0
MEMTOREG = X
MEMREAD = 1
ALUSRC = X
ALUOP = X-X
REGWRITE = 1

NUOVO SEGNALE :

LWIG = 1

200 + 50 + 150 + 200 + 50

2) Indicare il contenuto in bit della word che esprime l'istruzione
`lwig $s4, $t4, 0x0203`
compilando la tabella sottostante (assumiamo che lo *OpCode* di `wvss` sia `0x3D`)

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

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*.

PRIMA PASSA PER LA MEMORIA DELLE ISTRUZIONI (200 ns), POI, IN CONTEMPORANEA, SI PASSA PER I REGISTRI E PER UN SOMMATORE (PREVALE IL SOMMATORE), SI PASSA POI PER LA MEMORIA (200 ns) ED IL DATO PASSA PRIMA PER UN SOTTRATTORE (150 ns) E POI DI NUOVO NEI REGISTRI (50 ns)

Tempo = [MEM] + [SUM] + [MEM] + [SUB] + [R] + [R]

200 + 150 + 200 + 150 + 50 = 750 ns

ESSENDO CHE, IMPIEGA PIU' TEMPO RISPETTO L'ISTRUZIONE PIU' LUNGA (LW), PER AGGIUNGERE LWIG È NECESSARIO PROLUNGARE IL CICLO DI CLOCK.

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 MIPS modificata per supportare `Lwig` sia difettosa e sovrascriva il segnale `RegDst` come segue:
`RegDst` ← `1 - Jump` (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 `$v0` al termine dell'esecuzione del seguente frammento di codice, assumendo che i registri `$v0`, `$s0` e `$s1` siano inizializzati a `0`. Motivare la propria risposta.

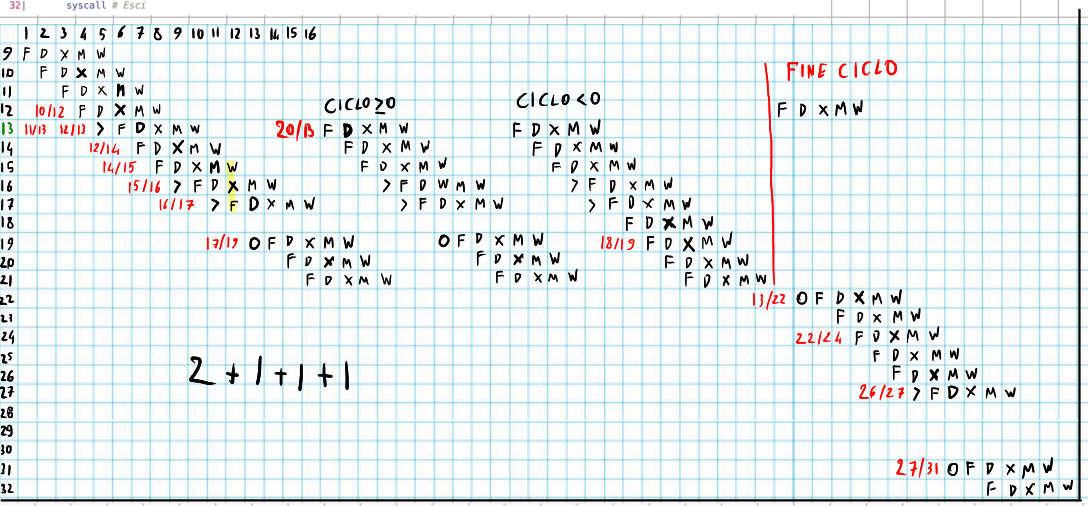
```
        addi $s0, $s0, 0x8004   REGDST = 1
        sub  $s1, $s0, $zero    REGDST = 1
        beq  $s1, $s0, Out      REGDST = 1
        j    Exit               REGDST = 0
Out:    add  $v0,  $0, $s1      REGDST = 1
Exit:   addi $v0, $v0, 0x8002   REGDST = 1
```

ADDI $S0, $S0, 0x8004 = XXXXXX 10000 10000 [15-11] 1000 0000 0000 0100 IN WRITE REGISTER FINIRA' IL REG.16 = $S0, QUINDI $S0 = 0x8004
SUB $S1, $S0, $ZERO FUNZIONA CORRETTAMENTE, QUINDI $S1 = 0x8004
BEQ $S1, $S0, OUT FUNZIONA CORRETTAMMENTE, IL BRANCH VIENE PRESO
OUT: ADD $V0, $0, $S1, FUNZIONA CORRETTAMENTE, QUINDI $V0 = 0x8004
EXIT: ADDI $V0, $V0, 0x8002 = XXXXXX 00010 00010 1000 0000 0000 0010 SCRIVE SU $S0 ALLA FINE SI HA $V0 = 0x8004

Esercizio 3 (11 punti)

Si consideri l'architettura MIPS con pipeline mostrata nella figura in basso (e in allegato). Il programma qui di seguito effettua la somma dei valori (in $s0) degli elementi presenti nell'array A di lunghezza L, dopo averne modificato i valori negativi moltiplicandoli per 2. Se il risultato è maggiore o uguale a 10, viene stampata la stringa S.

```
01| .globl main
02|
03| .data
04| A: .byte 0, 1, -1, -2, 0, -1
05| L: .byte 6
06| S: .asciiz "\nThe result is >= 10!"
07|
08| .text
09| 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|       sll  $s1, $s1, 1 # Altrimenti, moltiplica per due
19| sum:  add  $v0, $v0, $s1 # Somma $s1
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
```



M	N	S	M	N
10	12	V0	17	19
11	13	A2	13	22
12	13	A1	27	31
12	14	A1		
14	15	T1		
15	16	S1		
16	17	AT		
20	13	A1		
18	19	S1		
22	24	S0		
26	27	V1		

NO FORWARDING

	I	DH	CH	X	TOT
CARICAMENTO	4	0	0	1	4
PRE	4	2	0	1	7
C(≥0)	8	7	1	3	48
C(<0)	9	9	0	3	54
USCITA	1	1	0	1	2
POST	8	3	2	1	13
TOTALE					127

CON FORWARDING

	I	DH	CH	X	TOT
CARICAMENTO	4	0	0	1	4
PRE	4	1	0	1	5
C(≥0)	8	2	1	3	33
C(<0)	9	2	0	3	33
USCITA	1	0	0	1	1
POST	8	1	2	1	11
TOTALE					87

5) quali sono, per ognuna delle cinque fasi, le **istruzioni** (o le bolle) in pipeline durante il **12° ciclo di clock** (con **forwarding**);

IF: BEQ $AT, $ZERO, SUM
ID: BOLLA
EX: SLT $AT, $S1, $0
MEM: BOLLA
WB: LB $S1, ($t1)

6) in quale fase la decisione delle istruzioni di jump sia compiuta nel diagramma qui sotto, e quante fasi di stallo causi potenzialmente ogni control hazard.

La decisione di jump viene decisa in fase di Istruction Fetch, da come si può notare, durante questa fase, i bit dell'istruzione [31-26] finiscono in un comparatore (con 2) per poi finire come selettero del mux, se tali bit sono uguali a 2 (opCode dell'istruzione jump) il mux selezionerà appunto l'indirizzo indicato dai bit [25-0] , è quindi nella fase di Fetch che viene deciso che verrà eseguito un Jump.