Esercizio 1 (2020-10-27)

Sia data una CPU con processore a **4 GHz** e **8 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 **4 vie** con **8 set** e **blocchi da 4 word**; adopera una politica di rimpiazzo **LRU.**
L2 è una cache **direct-mapped** con **4 linee** e **blocchi da 16 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 ed indicare la metodologia di calcolo più in basso.

	Address	800	824	828	832	790	1640	836	848	1240	1244	1248	1252
L1	Block#	50	51	51	52	49	102	52	53	77	77	78	78
	Index	2	3	3	4	1	6	4	5	5	5	6	6
	Tag	6	6	6	6	6	12	6	6	9	9	9	9
	HIT/MISS	MISS	MISS	HIT	MISS	MISS	MISS	HIT	MISS	MISS	HIT	MISS	HIT
	Miss type	COLD	COLD		COLD	COLD	COLD		COLD	COLD		COLD	
L2	Block#	12	12		13	12	25		13	19		19	
	Index	0	0		1	0	1		1	3		3	
	Tag	3	3		3	3	6		3	4		4	
	HIT/MISS	MISS	HIT		MISS	HIT	MISS		MISS	MISS		HIT	
	Miss type	COLD			COLD		COLD		CONF	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 **200 ns**, che gli **hit** nella cache **L1** impieghino **2 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.

(2) L1

BLOCCO	128 +
V.bit	1 +
bit LRU	1 +
TAG	25 =
LINEA	155 ×
N° LINEE · VIE	32 =
TOTALE	496 bit

L2

BLOCCO	512 +
V.bit	1 +
TAG	24 =
LINEA	537 ×
N° LINEE	4 =
TOTALE	2148 bit

L1 + L2 = 2644 bit

(3)

TOT = 200 · 5 + 2 · 4 + 40 · 3 = 1128 ns

T. MEDIO = 1128 / 12 = 94 ns

COLPI MEDI = 94 · 4 = 376 C.C.

ISTR. MEDIE = 376 / 8 = 47 ISTRUZIONI

Esercizio 2 (2021-03-31)

Considerare l'architettura MIPS a ciclo singolo nella figura in basso (ed in allegato).
Si vuole aggiungere l'istruzione di tipo R "sum register values with a word in memory"
`sum_w_mem $rs, $rt, $rd, mem_addr`
(laddove mem_addr è indicato dal campo di 5 bit tipicamente assegnato allo `shamt`). L'istruzione deve salvare nel registro `rd` la somma di
1) `$rs` +
2) `$rt` +
3) la word in memoria all'indirizzo `mem_addr × 4 + 0x1000 0000` (il calcolo fa sì che l'indirizzo ricada nel *data segment*).

Supponiamo, per esempio, che il valore nel registro `rs` sia `0x40`, il valore nel registro `rt` sia `0x8`, e che `mem_addr` sia `0x13`. Dunque, la word da caricare sarà all'indice `0x13 × 4 + 0x1000 0000 = 0x1000 004C`; supponiamo che tale word in memoria abbia valore `0x7`. Il risultato della somma
`0x40 + 0x8 + 0x7 = 0x4F`
verrà salvato nel registro `rd`.

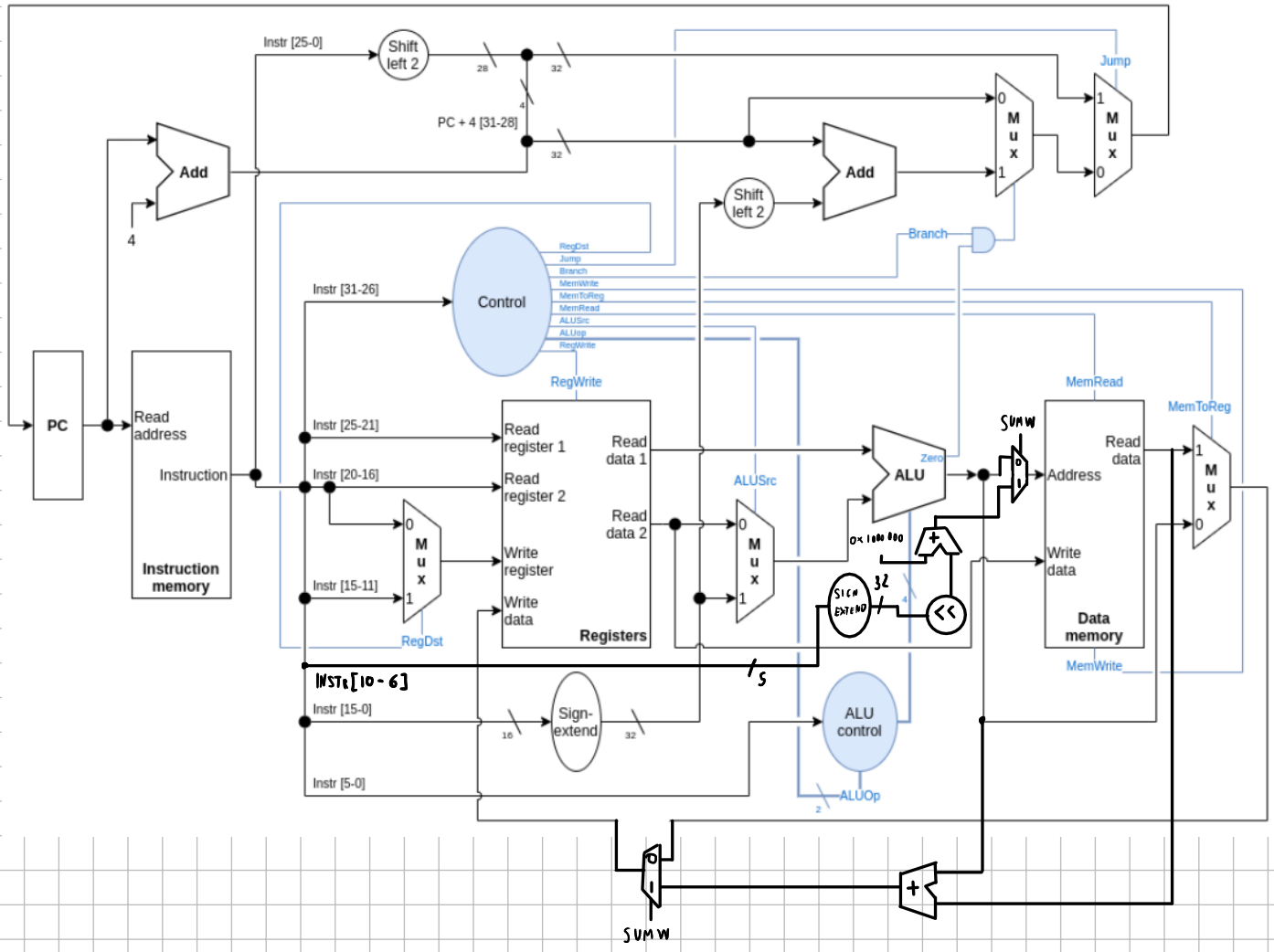
1) Mostrare le **modifiche all'architettura** della CPU MIPS, avendo cura di aggiungere eventuali altri componenti necessari a realizzare l'istruzione. A tal fine, si può alterare la stampa del diagramma architetturale oppure ridisegnare i componenti interessati dalla modifica, avendo cura di indicare 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
`sum_w_mem $a0, $s1, $t2, 0x1D`
compilando la tabella sottostante. Assumere che lo *OpCode* di `sum_w_mem` sia `0x33`. Nel campo `funct`, porre il sestetto di bit che indichi l'addizione.

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

3) Supponendo che l'accesso alle **memorie** impieghi **180 ns**, l'accesso ai **registri 60 ns**, le operazioni dell'**ALU e** dei **sommatori 120 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*.

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



(3) PRIMA SI PASSA PER LA MEMORIA ISTRUZIONI (180 ns), POI PER I REGISTRI (60 ns), POI PER L'ALU (120 ns), POI DI NUOVO PER LA MEMORIA (180 ns), DOPO DI CHE PER UN SOMMATORE (120 ns) ED INFINE PER I REGISTRI (60 ns) DATO IL WRITE BACK.

180 + 60 + 120 + 180 + 120 + 60 = 720 ns

È NECESSARIO PROLUNGARE IL CICLO DI CLOCK

Esercizio 3 (2020-09-11)

Considerare l'architettura MIPS a ciclo singolo nella figura in alto (ed in allegato). Si sospetta che la CPU abbia un **guasto**. In particolare, si ha il dubbio che la Control Unit sia malfunzionante e che pertanto produca il segnale di controllo **RegWrite** attivo se e solo se **non** è attivo il segnale **Jump**: RegWrite = *not*(Jump).
Si assume che:

– MemToReg = 1 solo per le istruzioni di tipo load, altrimenti valga 0 (ossia, mai "don't care", X)
– RegDst = 1 solo per le istruzioni di tipo R, altrimenti valga 0 (ossia, mai "don't care" X)
– MemRead = 1 solo per le istruzioni di tipo load, altrimenti valga 0 (ossia, mai "don't care" X)
– Branch = 1 solo per le istruzioni di salto condizionato, altrimenti valga 0 (ossia, mai "don't care" X)

1) Completare la tabella sottostante ipotizzando che ci sia il guasto. Per evidenziare i segnali errati a causa del guasto, aggiungere un punto esclamativo accanto: es., **1!** laddove il segnale corretto (ossia, senza guasto) sia 0, e **0!** laddove il segnale corretto sia 1.

	RegDst	ALUSrc	MemToReg	RegWrite	MemRead	MemWrite	Branch	Jump	ALUOp1	ALUOp0
Tipo R	1	0	0	1	0	0	0	0	**1**	**0**
lw	0	x	1	1	1	0	0	0	**0**	**0**
sw	0	x	0	1!	0	1	0	0	**0**	**0**
beq	0	x	0	1!	0	0	1	0	**0**	**1**
j	0	x	0	0	0	0	0	1	**X**	**X**

2) Realizzare un programma che scriva in **$s0** il valore **0xFFFFFFFF se il guasto occorre**. **Altrimenti**, $s0 deve contenere il valore **0**.
Spiegare con commenti il malfunzionamento atteso.

```
ADD  $S0, $ZERO, $ZERO // LE TIPO R FUNZIONANO CORRETTAMENTE
ADD  $S1, $S1, 1
BEQ  $S1, $0, END // NON SALTERA', MA ESSENDO MALFUNZIONANTE, SCRIVERA' SU $S0.
BEQ  $S0, $ZERO, FUNZIONA  //SE $S0 ≠ 0, È PERCHÈ IL BEQ HA ERRONEAMENTE
                           SCRITTO SU $S0 LA SOMMA
ADD  $S0, $ZERO, 0xFFFFFFFF  TRA $S0 ED $S1, OSSIA 1.
LI $V0, 10
SYSCALL
FUNZIONA:
ADD  $S0, $ZERO, $ZERO
LI $V0, 10
SYSCALL
```

Esercizio 4 (2021-01-29)

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 il conteggio degli elementi pari (in `$v0`) e dispari (in `$v1`) presenti nell'array `M` di lunghezza `N`.

```
.globl main

.data
M: .word 0, 151, 4885, 200, 19, 87 # Array
N: .byte 6                         # Lungh.
# Ris.: $v0 = 2 (el. pari); $v1 = 4 (dispari)
.text
main: la   $a0, M          # Indice base array
      lb   $a1, N          # Lunghezza array
      sll  $t7, $a1, 2     # $t7: last index…
      add  $t7, $t7, $a0   # … of the array
      sub  $v0, $t7, $t7   # Init. conto pari
      add  $v1, $v0, $v0   # Init. conto dispari
cycl: bge  $a0, $t7, exit  # Esci se $a0 >= $t7
      lw   $s0, ($a0)      # $s0: elem. array
      sll  $s0, $s0, 31    # $s0 << 31 (LSB)
      beq  $s0, $0, odd    # Se $s0 non è pari…
      addi $v1, $v1, 1     # … incrementa $v1…
      j    endC            # … altrimenti…
odd:  addi $v0, $v0, 1     # … incrementa $v0
endC: addi $a0, $a0, 4     # Avanza nell'array
      j cycl               # Ricomincia
exit: add $a0, $v0, $zero  # Carica $v0
      addi $v0, $zero, 1   # Imposta st. int.
      syscall              # Stampa $v0
      addi $v0, $zero, 11  # Imposta st. char.
      addi $a0, $0, 0x0A   # Carica linefeed
      syscall              # Stampa linefeed
      addi $v0, $zero, 1   # Imposta st. int.
      add $a0, $v1, $zero  # Carica $v1
      syscall              # Stampa $v1
      addi $v0, $zero, 10  # Esci
      syscall
```

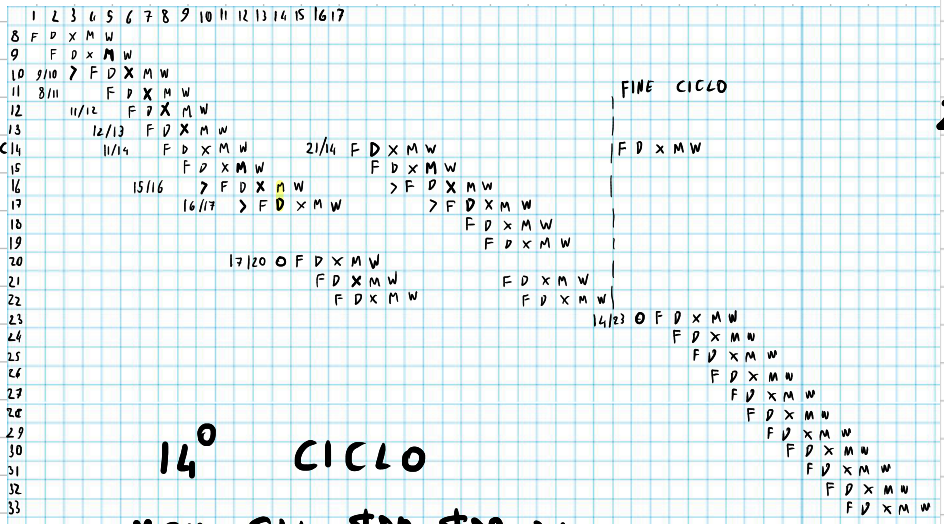
```
.globl main

.data
M: .word 0, 151, 4885, 200, 19, 87
N: .byte 6

.text
main: la   $a0, M
      lb   $a1, N
      sll  $t7, $a1, 2
      add  $t7, $t7, $a0
      sub  $v0, $t7, $t7
      add  $v1, $v0, $v0
cycl: bge  $a0, $t7, exit
      lw   $s0, ($a0)
      sll  $s0, $s0, 31
      beq  $s0, $0, odd
      addi $v1, $v1, 1
      j    endC
odd:  addi $v0, $v0, 1
endC: addi $a0, $a0, 4
      j cycl
exit: add $a0, $v0, $zero
      addi $v0, $zero, 1
      syscall
      addi $v0, $zero, 11
      addi $a0, $0, 0x0A
      syscall
      addi $v0, $zero, 1
      add $a0, $v1, $zero
      syscall
      addi $v0, $zero, 10
      syscall
```

LEGENDA:

STALLI PER DH = 2

STALLI PER CH = 0

DH			CH	
M	N	$	M	N
9	10	A1	17	20
8	11	A0	14	23
11	12	T7		
12	13	V0		
11	14	T7		
15	16	S0		
16	17	S0		
21	14	A0		



14^{0} CICLO

MEM = SLL $S0, $S0, 31

DECODE = BEQ $S0, $0, ODD

GLI ALTRI BOLLA

CON FORW.

	I	DH	CH	X	TOT
CARIC. P.	4	+ 0	+ 0	x 1	= 4 +
PRE CICL.	6	+ 1	+ 0	x 1	= 7 +
CICLO PARI	7	+ 2	+ 1	x 2	= 20 +
CICLO DISP.	8	+ 2	+ 0	x 4	= 40 +
USCITA	1	+ 0	+ 0	x 1	= 1 +
POST-CICLO	11	+ 0	+ 1	x 1	= 12 =
TOTALE					84

SENZA FORW.

	I	DH	CH	X	TOT
CARIC. P.	4	+ 0	+ 0	x 1	= 4 +
PRE CICL.	6	+ 4	+ 0	x 1	= 10 +
CICLO PARI	7	+ 5	+ 1	x 2	= 26 +
CICLO DISP.	8	+ 5	+ 0	x 4	= 52 +
USCITA	1	+ 1	+ 0	x 1	= 2 +
POST-CICLO	11	+ 0	+ 1	x 1	= 12 =
TOTALE					106