Esercizio 1 (11 punti).

#BL=ADR/128

Sia data una CPU con processore a **4GHz** 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 **2 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 32 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	2048	2040	2032	2024	2060	192	200	208	290	2048	2024	200
L1	Block#	256	255	254	253	257	24	25	26	36	256	253	25
	Index	0	3	2	1	1	0	1	2	0	0	1	1
	Tag	64	63	63	63	64	6	6	6	9	64	63	6
	HIT/MISS	MISS	MISS	MISS	MISS	MISS	MISS	MISS	MISS	MISS	MISS	MISS	HIT
	Miss type	COLD	COLD	COLD	COLD	COLD	COLD	COLD	COLD	COLD	CONF	CONF	
L2	Block#	16	15	15	15	16	1	1	1	2	16	15	
	Index	0	3	3	3	0	1	1	1	2	0	3	
	Tag	4	3	3	3	4	0	0	0	0	4	3	
	HIT/MISS	MISS	MISS	HIT	HIT	HIT	MISS	HIT	HIT	MISS	HIT	HIT	
	Miss type	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 sesto indirizzo (192) 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 **1KiB** ciascuna, indicarne i numeri di pagina virtuale. Si assume che la cache sia a monte della memoria virtuale.

Address	2048	2040	2032	2024	2060	192	200	208	290	2048	2024	200
Page#												

(2) L1 = $4 \cdot 2 \cdot (64 + 1 + 1 + 28) = 752$ bit

L2 = $4 \cdot (1024 + 1 + 23) = 4192$ bit

64 = blocco
1 = V.bit
1 = LRU
28 = tag

1024 = blocco
1 = V. bit
23 = tag

(3)

TOT = $4 \cdot 400 + 1 + 7 \cdot 40 = 1881$ ns

TEMPO MEDIO = 156.75 ns

4GHz = 4 C.C. OGNI 1 ns → COLPI MEDI = 627 C.C.

OGNI ISTR. = 4 C.C. → ISTRUZIONI IN ACC. MEDIO = 156.75

(4)

192 % 128 = 64

$\frac{64}{4} = 16$ ← WORD OFFSET

Esercizio 2 (11 punti).

Considerare l'architettura MIPS a ciclo singolo nella figura in basso (ed in allegato).
Si vuole aggiungere alla CPU l'istruzione **load parity keyword** (`pkl`), di **tipo R** e sintassi assembly
`pkl $op_reg1, $op_reg2, $out_reg, select`
dove `op_reg1`, `op_reg2`, `out_reg` e `select` sono rispettivamente nei campi `rs`, `rt`, `rd` e `shamt` dell'istruzione:
1) seleziona dal valore del registro `$op_reg1` il bit di indice `select`
2) seleziona dal valore del registro `$op_reg2` il bit di indice `select`;
3) carica in registro `$out_reg` una word che è registrata in memoria ad un indirizzo che dipende dal risultato della somma dei bit selezionati ai punti (1) e (2):
3.1) se il risultato della somma è pari, carica in `out_reg` la parola che è registrata in memoria all'indirizzo **`0x1001 1010`**;
3.1) se il risultato della somma è dispari, carica in `out_reg` la parola che è registrata in memoria all'indirizzo **`0x1001 1014`**.

Esempio: Supponiamo che `$op_reg1` sia `$s0`, contenente `0x0F0F 0F0A` = 0000 1111 0000 1111 0000 1111 0000 1010$_{due}$,
che `$op_reg2` sia `$s1`, contenente `0x0FDF 070A` = 0000 1111 1101 1111 0000 0111 0000 1010$_{due}$,
che `$out_reg` sia `$v0`,
che `select` sia 8.
La somma dei bit selezionati (in grigio in alto, all'indice 8) è pari. Dunque, immaginando che all'indirizzo in memoria `0x1001 1010` sia registrata la word "Even" (ossia `0x6e657645` in MIPS) in `$v0` viene salvato `0x6e657645`.

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*.

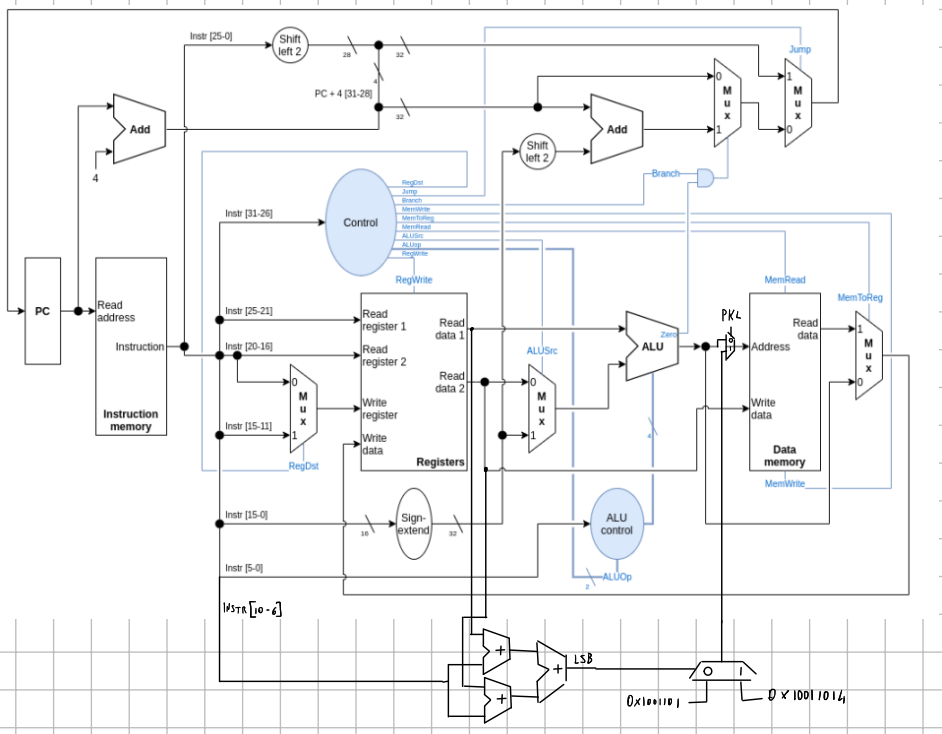
2) Indicare il contenuto in bit della word che esprime l'istruzione
`pkl $s0, $s1, $v0, 9`
compilando la tabella sottostante (assumiamo che lo *OpCode* di `pkl` sia `0x3D` mentre il campo *func* sia `0x26`)

1	0	0	1	1	0															0	1	0	0	1	x	x	x	x	x	x

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 `Pkl` 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 `Pkl` sia difettosa e sovrascriva il segnale `ALUSrc` come segue:
`ALUSrc` ← `Pkl` (il simbolo ← denota che la variabile a sinistra assume il valore dato della variabile a destra)
In tal caso, indicare quale valore venga salvato in memoria e a quale indirizzo al termine dell'esecuzione del seguente frammento di codice, assumendo che i registri `$s0` e `$s1` siano inizializzati a `0`. Motivare la propria risposta.

REGDST	JUMP	BRANCH	MEMWRITE	MEMTOREG	MEMREAD	ALUSRC	ALUOP	REGWRITE	PKL
1	0	0	0	1	1	x	x x	1	1



MEM INS DEC ALU ALU MEM WB

$t = 200 + 50 + 150 + 150 + 200 + 50$

$= 800$ ms

Esercizio 3 (11 punti)

Si consideri l'architettura MIPS con pipeline mostrata nella figura in basso (e in allegato). Il programma qui di seguito conta il numero di occorrenze pari e dispari strettamente maggiori di zero, usando il numero -1 come byte di uscita.

Si noti che, dato l'input fornito, il codice conterà un'occorrenza pari e due occorrenze dispari (maggiori di 0).

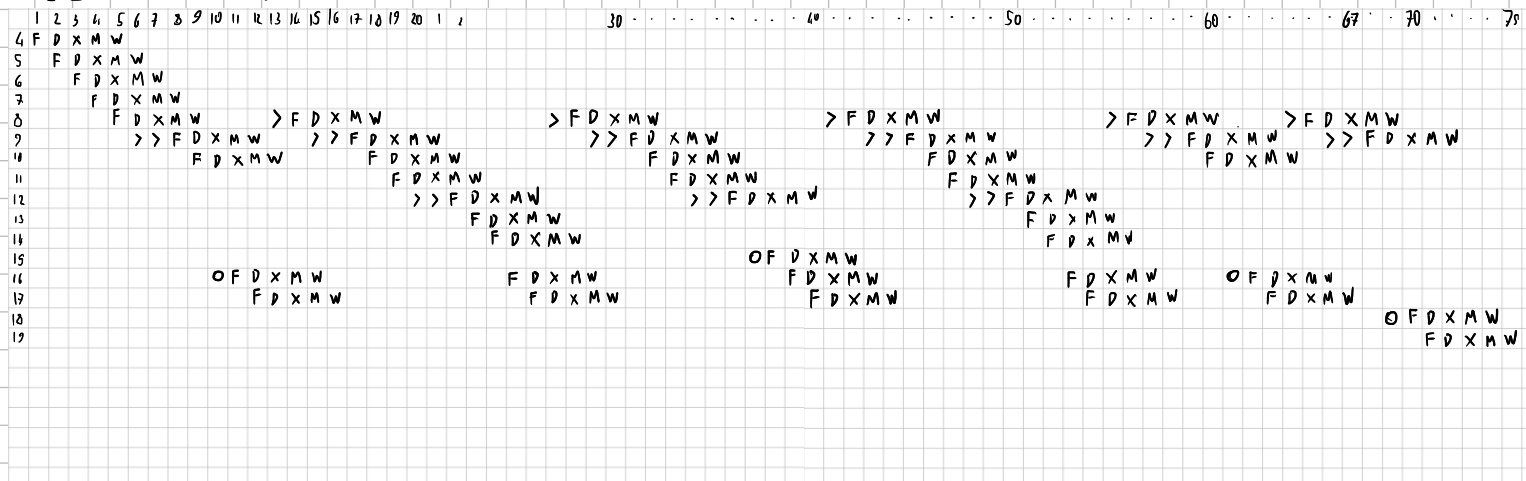
```
1   .data
2           S: .byte 0,7,8,9,0,-1
3   .text # $v0: somma occorrenze pari > 0, $v1: somma occorrenze dispari > 0
4   main:   la $a0, S                   # Carica indirizzo array
5           li $v0, 0                   # Inizializza $v0
6           li $v1, 0                   # Inizializza $v1
7           li $t1, -1                  # byte di uscita
8   cycle:  lb  $s0, ($a0)              # Carica byte
9           beq $s0, $t1, exit          # È il byte di uscita? Esci
10          beq $s0, $zero, endIf       # Non lo conta se è uguale ad 0
11          andi $s0, $s0, 1            # $s0 uguale a 1 se $s0 è dispari
12          beq $s0,$zero, pari         # Salta la prossima istruzione se è pari
13          addi $v1, $v1, 1            # Somma contatore 1
14          j endIf
15  pari:   addi $v0, $v0, 1            # Somma contatore 0
16  endIf:  addi $a0, $a0, 1            # Incrementa di 1 byte il contatore array
17          j    cycle                  # Ripeti il ciclo
18  exit:   addi $v0, $zero, 10         # Imposta uscita
19          syscall
```

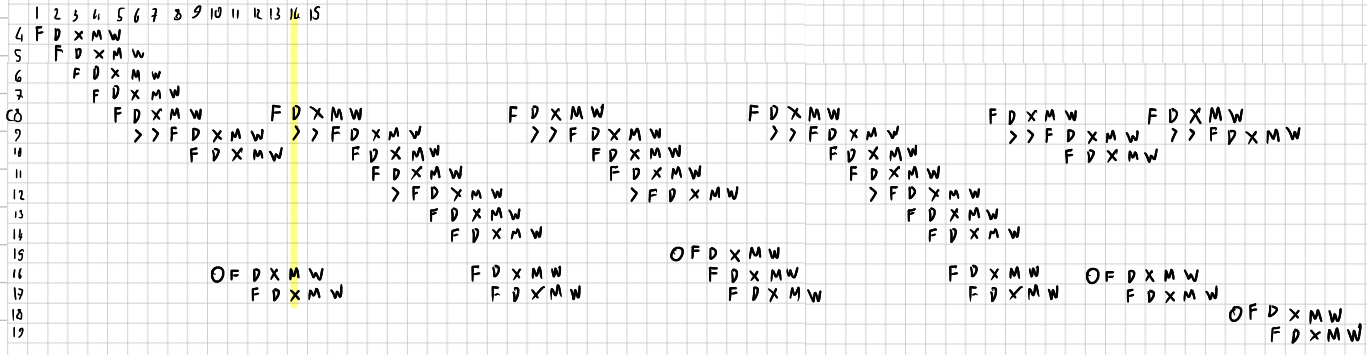
Si supponga che *tutte le istruzioni impiegate facciano parte del set supportato dalla CPU in figura*, ossia non si fa uso di alcuna pseudoistruzione.

Si indichino (ignorando hazard che possano concernere la syscall):
1) le istruzioni tra le quali sono presenti **data hazard** – indicare i numeri di istruzione N ed M (visibili alla sinistra delle linee di codice in figura) ed il registro coinvolto $xY (nel formato N / M / $xY);

2) le istruzioni tra le quali sono presenti **control hazard** – indicare i numeri di istruzione N ed M (nel formato N / M);

SENZA FORWARDING





5) quali sono, per ognuna delle cinque fasi, le **istruzioni** (o le bolle) in pipeline durante il **14° ciclo di clock** (con **forwarding**);

IF: BOLLA
ID: LB $s0, ($a0)
EX: J CYCLE
MEM: ADDI $a0, $a0, 1
WB: BOLLA

NO FORWARDING

	I	DH	CH	X	TOT
	4	0	0	1	4
	4	0	0	1	4
CD	9	5	0	2	28
CP	8	5	1	1	14
CO	5	3	1	2	17
UC	2	3	0	1	5
PC	2	0	1	1	3
					75

FORWARDING

	I	DH	CH	X	TOT
	4	0	0	1	4
	4	0	0	1	4
CD	9	3	0	2	24
CP	8	3	1	1	12
CO	5	2	1	2	16
UC	2	2	0	1	4
PC	2	0	1	1	3
					67

DATA HAZARD

M	N	$
8	9	$50
16	8	$20
11	12	$50

M	N
10	16
12	15
9	17