Esercizio 1 (11 punti).

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#**												
	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 **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#												

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`.

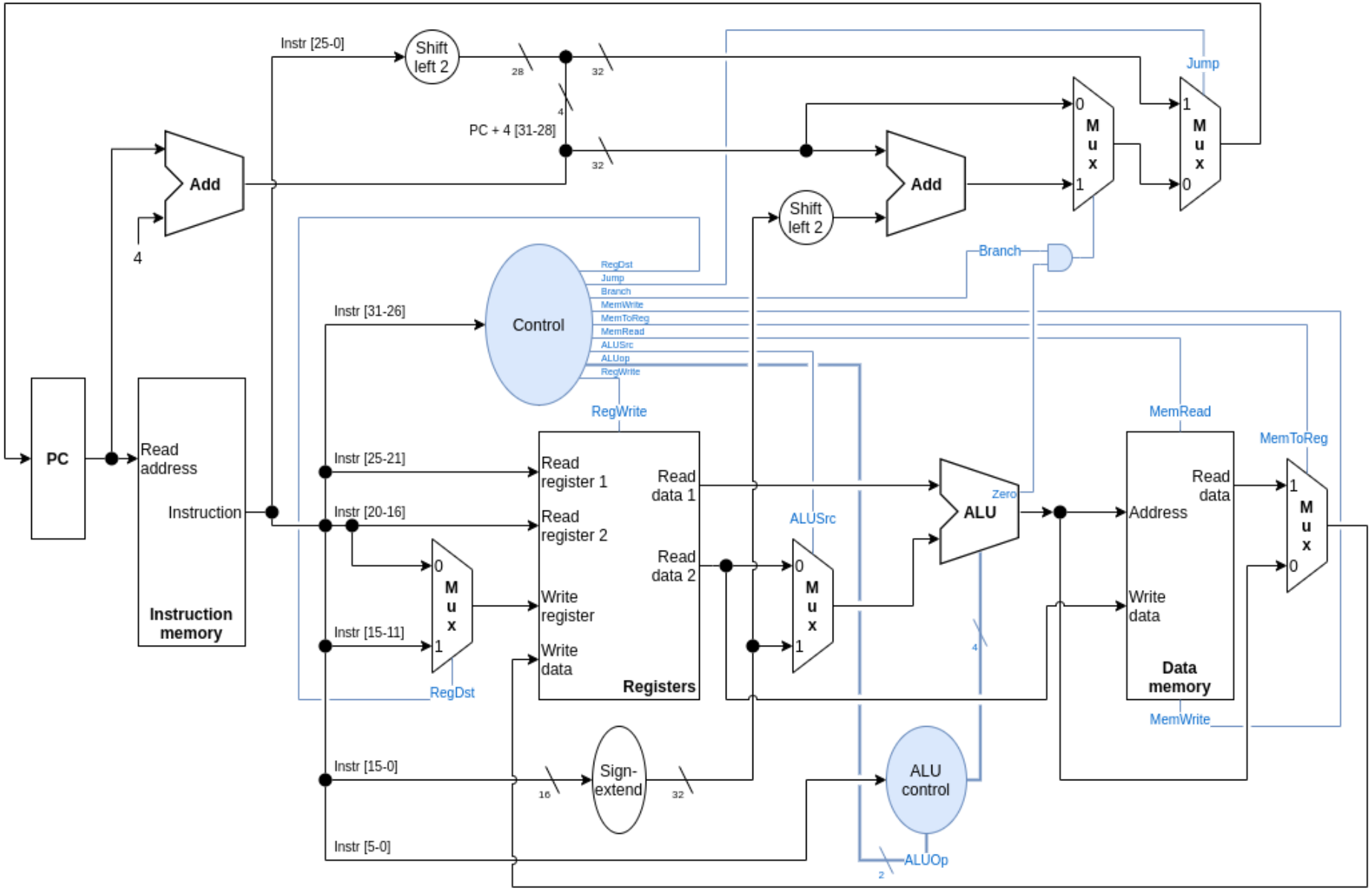
1) Mostrare le **modifiche all'architettura** della CPU MIPS, avendo cura di aggiungere eventuali altri componenti necessari a realizzare l'istruzione. A tal fine, si può alterare la stampa del diagramma architetturale oppure ridisegnare i componenti interessati dalla modifica, avendo cura di indicare 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`)

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.

```
     addi $s1,  $0, 0x4
     beq  $s1, $s0, Out
     add  $s0, $s0, $s1
Out: addi $s1, $s1, 0x006F
     sw   $s0, ($s1)
```



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);

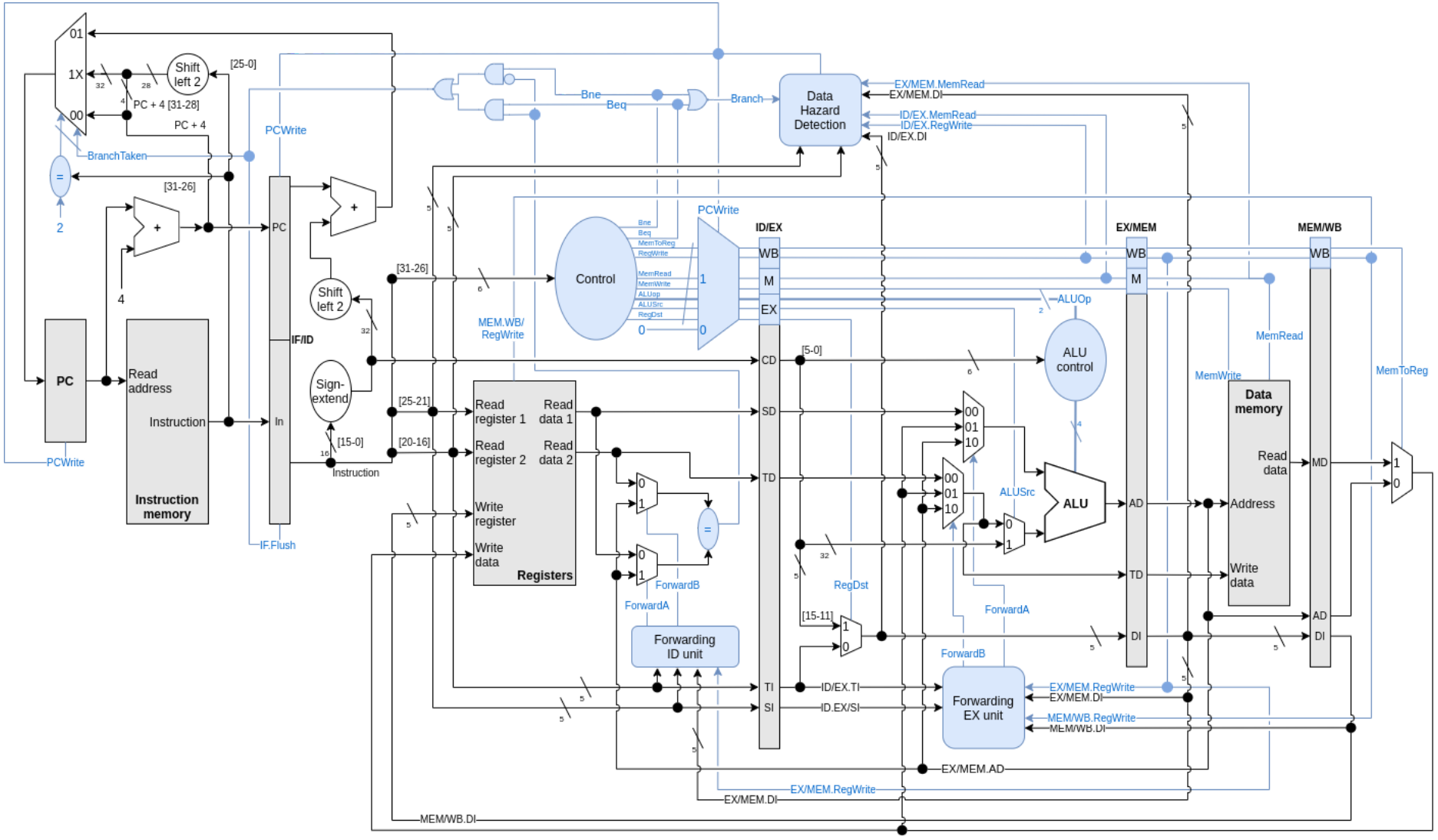
3) quanti **cicli di clock** sono necessari ad eseguire il programma tramite **forwarding**, spiegando il calcolo effettuato;

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 **14° ciclo di clock** (con **forwarding**);

IF:
ID:
EX:
MEM:
WB:

6) in quale fase della pipeline venga presa la decisione dell'istruzione *jump***,** e quali conseguenze questo tipo di architettura abbia sugli hazard (si ricordi di consultare l'immagine dell'architettura seguente questa domanda)


01
1X
00
Shift left 2
[25-0]
32
28
4
PC + 4 [31-28]
PC + 4
PCWrite
BranchTaken
=
[31-26]
2
+
PC
IF/ID
In
Shift left 2
Sign-extend
[15-0]
16
Instruction
IF.Flush
PC
PCWrite
Read address
Instruction
Instruction memory
Bne
Beq
Branch
Data Hazard Detection
EX/MEM.MemRead
EX/MEM.DI
ID/EX.MemRead
ID/EX.RegWrite
ID/EX.DI
Control
Bne
Beq
MemToReg
RegWrite
MemRead
MemWrite
ALUop
ALUSrc
RegDst
PCWrite
ID/EX
WB
M
EX
CD
SD
TD
TI
SI
MEM.WB/ RegWrite
[31-26]
[25-21]
[20-16]
[5-0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
ForwardB
ForwardA
Forwarding ID unit
ALU control
ALUOp
ALUSrc
ALU
RegDst
[15-11]
ID/EX.TI
ID.EX/SI
Forwarding EX unit
EX/MEM.RegWrite
EX/MEM.DI
MEM/WB.RegWrite
MEM/WB.DI
EX/MEM.AD
EX/MEM
AD
TD
DI
MemRead
MemWrite
Data memory
Read data
Address
Write data
MEM/WB
MD
MemToReg
MEM/WB.DI