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#**												
	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. 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#												

Nome, Cognome, Matricola:_____ **Firma:**_____

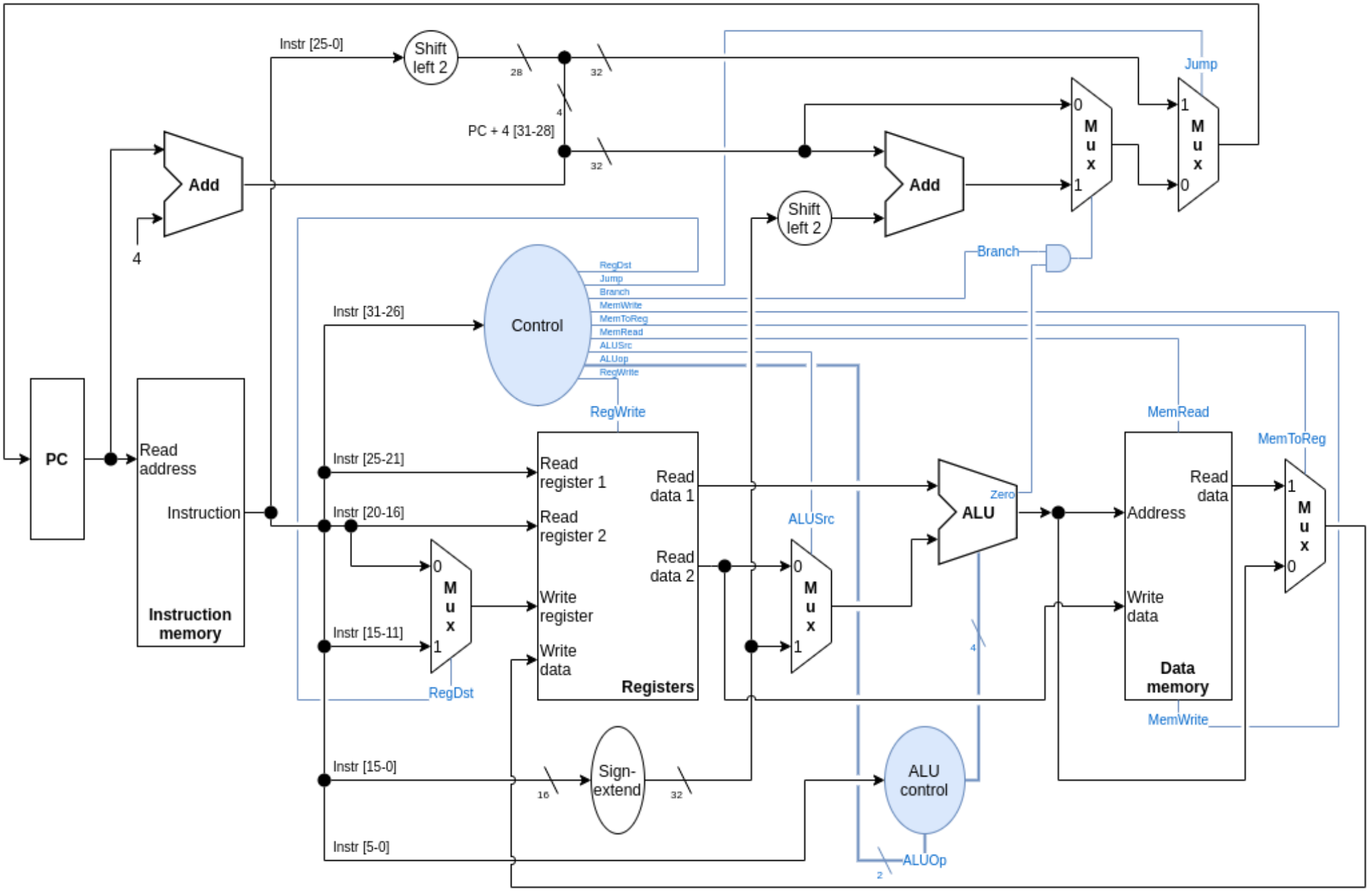
Esercizio 2 (11 punti).

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*.
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`)

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


Instr [25-0]
Shift left 2
28
32
4
PC + 4 [31-28]
32
Add
4
Add
Shift left 2
Add
0
M
u
x
1
1
M
u
x
0
Jump
Branch
RegDst
Jump
Branch
MemWrite
MemToReg
MemRead
ALUSrc
ALUop
RegWrite
Control
Instr [31-26]
RegWrite
MemRead
MemToReg
PC
Read address
Instruction
Instruction memory
Instr [25-21]
Instr [20-16]
Instr [15-11]
Instr [15-0]
Instr [5-0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
RegDst
ALUSrc
Zero
ALU
Address
Read data
Write data
Data memory
MemWrite
Sign-extend
16
32
ALU control
ALUOp
2
4

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
```

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);

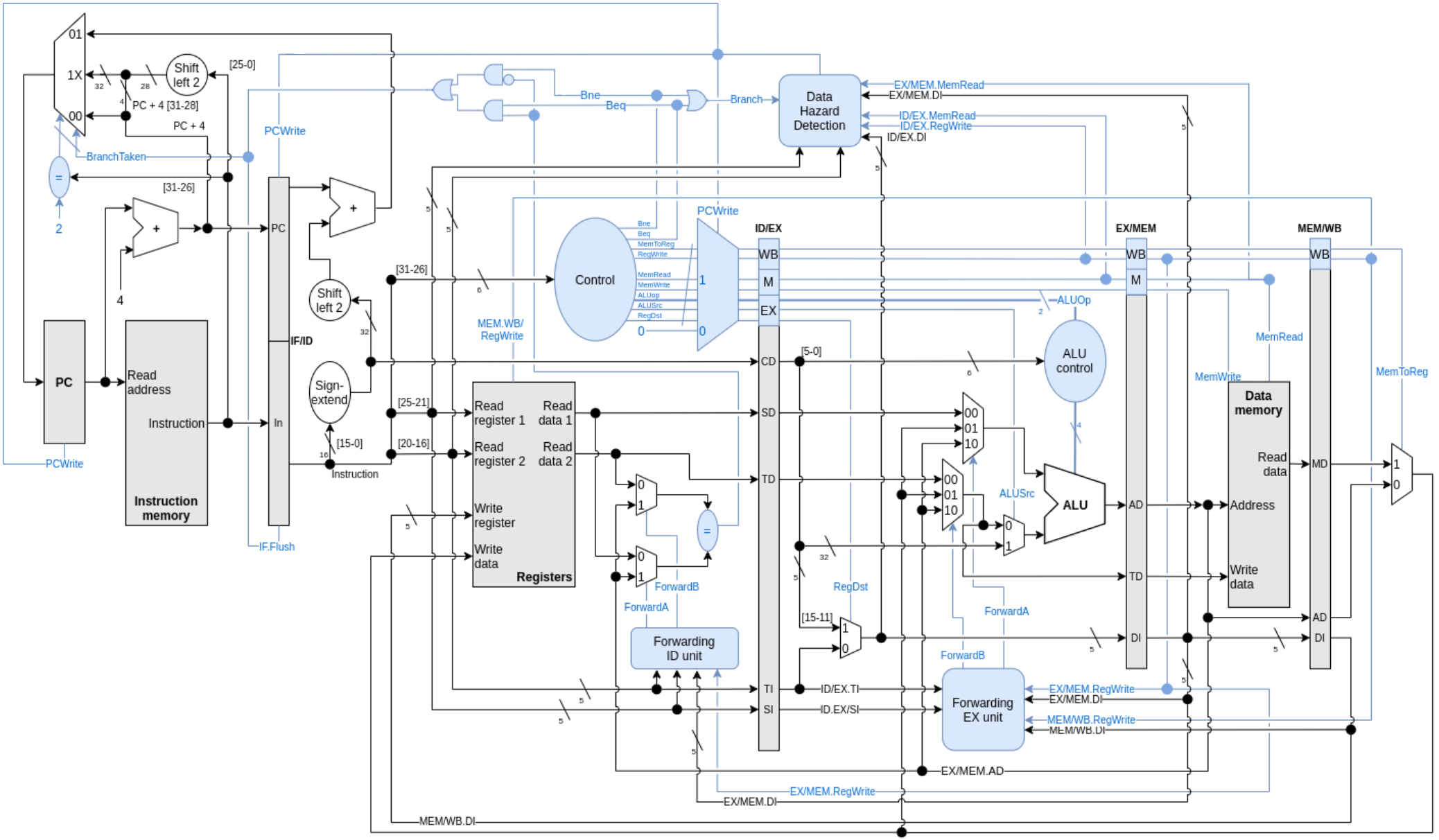
2) le istruzioni tra le quali sono presenti **control hazard** – indicare i numeri di istruzione N ed M (nel formato N / M);

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 **12° ciclo di clock** (con **forwarding**);
   IF:
   ID:
   EX:
   MEM:
   WB:

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.


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