Esercizio 1 (11 punti). 0 16 - 20 1 1 - 5 2 3 27 -

Sia data una CPU con processore a **2GHz** 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 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	128	8	162	40	8000	192	12	10	220	130	160	480
L1	Block#	16	1	20	5	1000	24	1	1	27	16	20	60
	Index	0	1	0	1	0	0	1	1	3	0	0	0
	Tag	4	0	5	1	250	6	0	0	6	4	5	15
	HIT/MISS	MISS	MISS	MISS	MISS	MISS	MISS	HIT	HIT	MISS	MISS	MISS	MISS
	Miss type	COLD	COLD	COLD	COLD	COLD	COLD			COLD	CONF	CONF	COLD
L2	Block#	2	0	2	0	125	3			3	2	2	7
	Index	2	0	2	0	1	3			3	2	2	3
	Tag	0	0	0	0	31	0			0	0	0	1
	HIT/MISS	MISS	MISS	HIT	HIT	MISS	MISS			HIT	HIT	HIT	MISS
	Miss type	COLD	COLD			COLD	COLD						COLD

L1 : #BL = ADR / 8 IND = #BL % 4 TAG = #BL / 4

L2 : #BL = ADR / 64 IND = #BL % 4 TAG = #BL / 4

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.

L1 VIE x LINEE x (BLOCCO + V.bit + LRU + TAG)

2 · 4 · (64 + 1 + 1 + 28) = 752 bit

L2 LINEE x (BLOCCO + V.bit + TAG)

4 · (512 + 1 + 24) = 2148 bit

L1 + L2 = 2900 bit

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.

TEMPO TOTALE : 400 · 5 + 1 · 2 + 40 · 5 = 2202 ns

TEMPO MEDIO : 2202 / 12 = 183.5 ns

COLPI MEDI : 183.5 · 2 = 367 C.C.

ISTRUZIONI MEDIE: 367 / 4 = 91.75 ISTR.

4) Calcolare il word offset del quinto indirizzo (8000) per la cache L1 spiegando i calcoli effettuati.

$$\text{WORD OFFSET} = \left\lfloor \frac{\text{ADR \% BYTE x BLOCCO}}{4} \right\rfloor = \left\lfloor \frac{8000 \% 8}{4} \right\rfloor = \frac{0}{4} = 0$$

5) Considerando i tipi di miss che possono verificarsi durante l'uso della memoria virtuale, indicare se possa presentarsi il caso in cui si abbiano sia *cache hit* sia *page table miss*, spiegando brevemente il motivo. Si noti che questa è una domanda teorica non collegata alla domanda (1) di questo esercizio.

È IMPOSSIBILE, DATO CHE LA PAGE TABLE, SE DA UN MISS, INDICA CHE QUEL DATO/PAGINA NON È PRESENTE IN MEMORIA, QUINDI NON PUO' A PRIORI STARE NELLA CACHE.

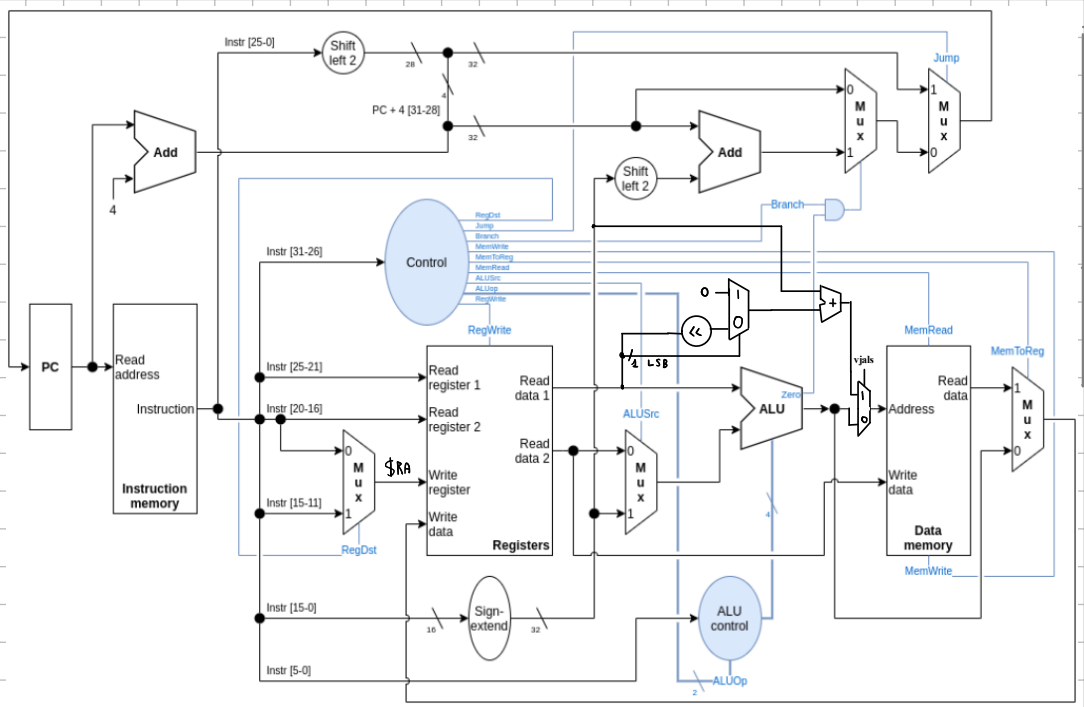
Esercizio 2 (11 punti).

Considerare l'architettura MIPS a ciclo singolo nella figura in basso (ed in allegato).
Si vuole aggiungere alla CPU l'istruzione vectorised jump and link safe (vjals), di tipo I e sintassi assembly
`vjals $indice, vettore`
che salta all'indirizzo contenuto nell'elemento $indice-esimo del vettore di word e salva il valore PC+4 in `$ra` (si ricorda che `$ra` è l'ultimo registro). Inoltre se il valore $indice è minore di zero, l'istruzione sceglierà il primo elemento (0-simo) del vettore.

Esempio 1: se in memoria si è definito staticamente il vettore:
`.word 15, 24, 313, 42`
e nel registro $t0 c'è il valore 3 allora
`vjals $t0, vettore`
salterà all'indirizzo 42 (che è l'elemento con indice 3 del vettore) e salverà il valore di PC+4 in $ra.

Esempio 2: Se invece nel registro `$t0` c'è il valore `-8` l'istruzione salterà all'indirizzo 15 (che è l'elemento con indice 0 del vettore) e salverà il valore di PC+4 in $ra.

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	1
MEMTOREG	1
MEMREAD	1
ALUSRC	x
ALUOP	x - x
REGWRITE	1

VJALS = 1

2) Indicare il contenuto in bit della word che esprime l'istruzione
`vjals $16, 0x0FFA`
compilando la tabella sottostante (assumiamo che lo *OpCode* di `vjals` sia `0x38`, e si ricorda che `$ra` è l'ultimo registro).

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

3) Supponendo che l'accesso alle **memorie** impieghi **75 ns**, l'accesso ai **registri 25 ns**, le operazioni dell'**ALU** e dei **sommatori 100 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*.

MEM 75MS | REG 25MS | SUM 100 MS | MEM 75MS | REG 25MS = 300 MS

4) Indicando con `vjals` 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 `vjals` sia difettosa e sovrascriva il segnale `RegDst` come segue:
`RegDst = vjals`
In tal caso, quale sarà il valore di $2 al termine dell'esecuzione del seguente frammento di codice, assumendo che $s2 e il PC siano **inizializzati a 0**? Spiegare il motivo.

```
addi    $2, $0, -1
vjals   $2, 0x0100
```

ADDI NON FUNZIONERA' CORRETTAMENTE, DATO CHE VJALS = 0 E REGDST NELLE ISTRUZIONI DI TIPO I DOVREBBE VALERE 1 MA VARRA' 0, QUINDI IL REGISTRO DI SCRITTURA SARA' QUELLO DEFINITO DAI BIT [20-16] OSSIA $0. IN $S2 NON SARA' SCRITTO IL DATO -1. VJALS DEVE AVERE REGDST = 0 MA SARA' 1, QUINDI IL RISULTATO SARA' SCRITTO IN [15-11], OSSIA NEL REGISTRO 00000.
DATO CHE L'ISTR. SARA': 111000 00010 11111 00000001 00000000. AL TERMINE $S2 = 0.
[15-11]

Esercizio 3 (11 punti)

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 le operazioni di somma o sottrazione indicate nella stringa fornita in input.

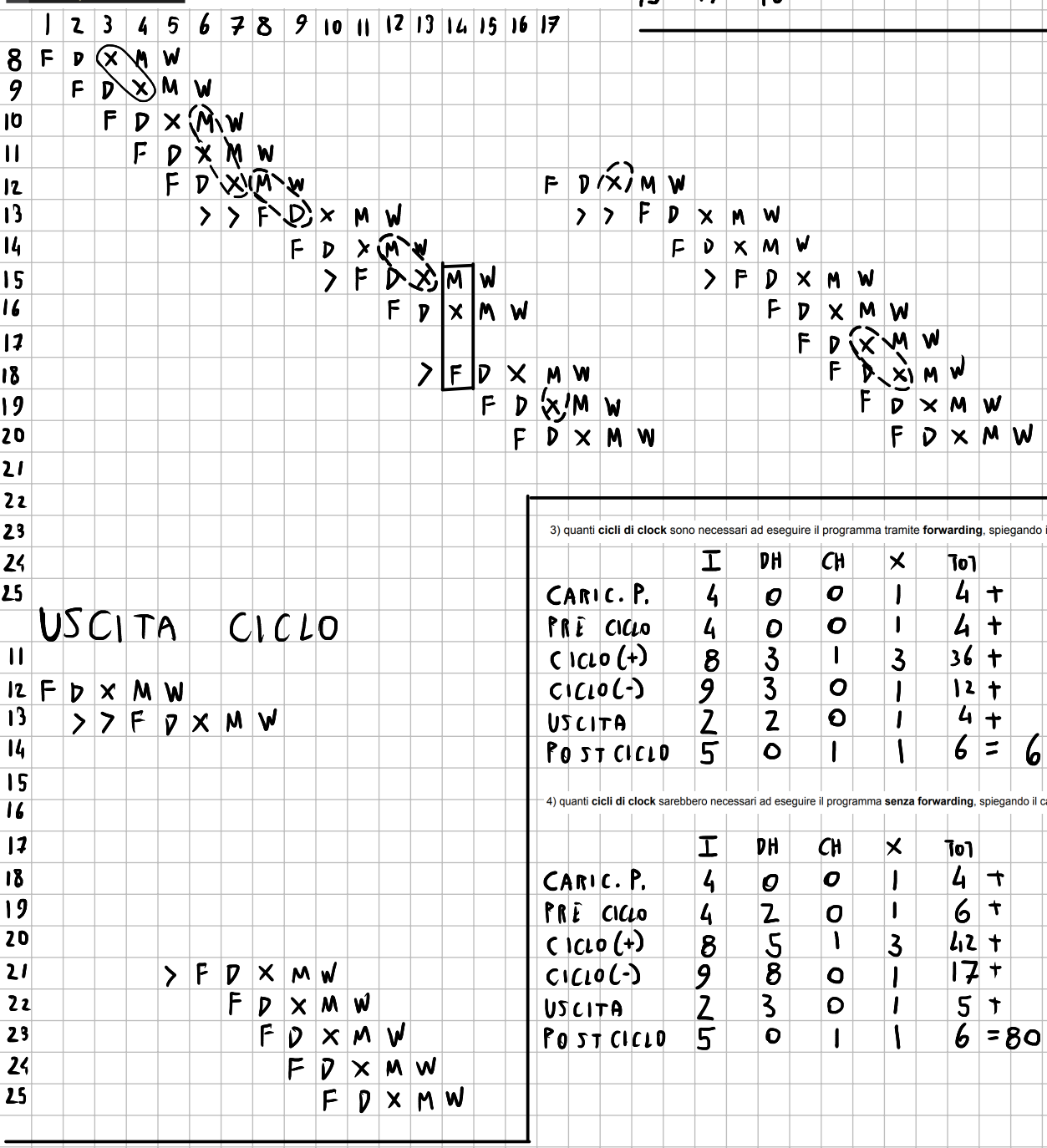
```
1  .globl main
2  .data
3  W: .asciiz "+2-3+4+2"
4
5  .text
6
7  main:
8          and $t1, $t0, $t0
9          and $t0, $t0, $zero
10         la $a0, W
11         or $v0, $zero, $zero
12 Cycle:  lb $t1, 0($a0)
13         beq $t1, $zero, Exit
14         lb $t0, 1($a0)
15         sub $t0, $t0, '0'
16         bne $t1, '-', Add
17         sub $t0, $zero, $t0
18 Add:    add $v0, $v0, $t0
19         addi $a0, $a0, 2
20         j Cycle
21 Exit:   move $a0, $v0
22         li $v0, 1
23         syscall
24         li $v0, 10
25         syscall
```

```
.globl main

.data
W: .asciiz "+2-3+4+2"

.text
main:
        and  $t0, $t0, $zero
        and  $t1, $t0, $t0
        la   $a0, W
        or   $v0, $zero, $zero
Cycle:  lb   $t1, 0($a0)
        beq  $t1, $zero, Exit
        lb   $t0, 1($a0)
        subi $t0, $t0, '0'
        bne  $t1, '-', Add
        sub  $t0, $zero, $t0
Add:    add  $v0, $v0, $t0
        addi $a0, $a0, 2
        j    Cycle
Exit:   move $a0, $v0
        li   $v0, 1
        syscall
        li   $v0, 10
        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);

DATA HAZARD			CONTROL HAZARD	
M	N	$	M	N
8	9	T0	16	18
10	12	A0	13	21
12	13	T1		
14	15	T0		
17	18	T0		
19	12	A0		
15	17	T0		



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

	I	DH	CH	X	Tot	
CARIC. P.	4	0	0	1	4	+
PRE CICLO	4	0	0	1	4	+
CICLO (+)	8	3	1	3	36	+
CICLO (-)	9	3	0	1	12	+
USCITA	2	2	0	1	4	+
POST CICLO	5	0	1	1	6	= 66 COLPI

4) quanti **cicli di clock** sarebbero necessari ad eseguire il programma **senza forwarding**, spiegando il calcolo effettuato;

	I	DH	CH	X	Tot	
CARIC. P.	4	0	0	1	4	+
PRE CICLO	4	2	0	1	6	+
CICLO (+)	8	5	1	3	42	+
CICLO (-)	9	8	0	1	17	+
USCITA	2	3	0	1	5	+
POST CICLO	5	0	1	1	6	= 80 COLPI

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

IF: ADD $v0, $v0, $t0
ID: BOLLA
EX: BNE $t1, '-', ADD
MEM: SUB $t0, $t0, 0
WB: BOLLA

6) qualora la decisione delle istruzioni di jump sia compiuta in fase ID, quante fasi di stallo causi potenzialmente ogni control hazard. Chiarire se questo cambiamento abbia altri effetti.

Ad ogni jump verrebbe causato un control hazard dato dalla politica del branch not taken, dato che, se il salto viene preso in fase di ID, verrà caricata la fase di IF della prossima istruzione.

(a questa domanda ho risposto a caso)